University
of Southern
California

Collected Papers of the Soar/IFOR Project, Spring 1995

W. Lewis Johnson, Randolph M. Jones, Frank V. Koss,
John E. Laird, Jill F. Lehman, Paul E. Nielsen,
Paul S. Rosenbloom, Robert Rubinoff, Karl B. Schwamb,
Milind Tambe, Julie Van Dyke, Michael van Lent, and Robert E. Wray III

USC/Information Sciences Institute, University of Michigan
and
Carnegie Mellon University

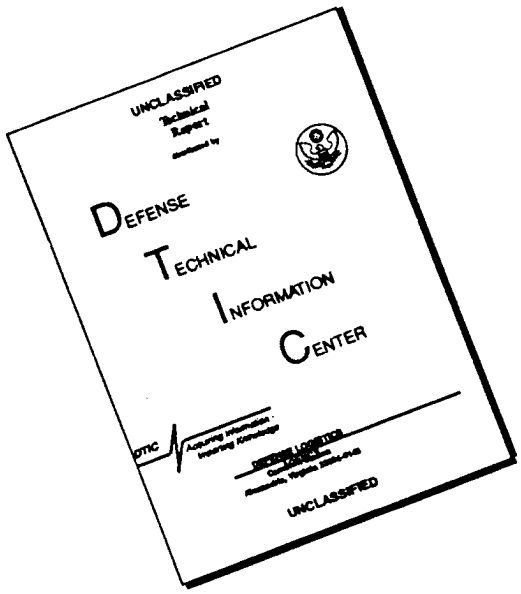
June 1995

ISI/SR-95-406

INFORMATION
SCIENCES
INSTITUTE

310/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1995	Research Report

4. TITLE AND SUBTITLE

Collected Papers of the Soar / IFOR Project, Spring 1995

5. FUNDING NUMBERS

N00014-92-K-2015
and
N66001-95-C-6013

6. AUTHOR(S)

W. Lewis Johnson, Randolph M. Jones, Frank V. Koss, John E. Laird, Jill F. Lehman, Paul E. Nielsen, Paul S. Rosenbloom, Robert Rubinoff, Karl B. Schwamb, Milind Tambe, Julie Van Dyke, Michael van Lent, and Robert E. Wray III

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER

ISI/SR-95-406

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

ARPA/ ASTO
3701 N. Fairfax Drive
Arlington, VA 22203-1714

Naval Research Laboratory
4555 Overlook Ave., SW
Washington, D.C. 20375-5000

NRaD / NCCOSC
53560 Hull St.
San Diego, CA 92152

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT

UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Since the summer of 1992, the Soar/IFOR research group has been building intelligent automated agents for tactical air simulation. The Soar/IFOR research project exists at three sites, the University of Michigan, the University of Southern California, and Carnegie Mellon University. The ultimate goal of this project is to develop automated pilots whose behavior in simulated engagements is indistinguishable from that of human pilots. Our work has concentrated on developing agents for a variety of air-to-air and air-to-ground missions.

This technical report is a collection of research papers that have been generated from this project between Spring 1994 and Spring 1995. The research covered in these papers spans a wide spectrum of issues in agent development such as learning, planning, coordination, command and control, natural language processing, agent tracking, and piloting rotary wing aircraft.

14. SUBJECT TERMS

Battlefield simulation, intelligent agents, intelligent computer-generated forces, interactive simulation, machine learning, military, Soar, tactical air simulation

15. NUMBER OF PAGES

78

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Table of Contents

Preface

Since the summer of 1992, the Soar/IFOR research group has been building intelligent automated agents for tactical air simulation. The Soar/IFOR research project exists at three sites, the University of Michigan, the University of Southern California, and Carnegie Mellon University. The ultimate goal of this project is to develop automated pilots whose behavior in simulated engagements is indistinguishable from that of human pilots. Our work has concentrated on developing agents for a variety of air-to-air and air-to-ground missions.

This technical report is a collection of the research papers that have been generated from this project between Spring 1994 and Spring 1995. Earlier papers were published as "Collected papers of the Soar/IFOR project, Spring 1994", Johnson, W. L, et al., Technical Reports CSE-TR-207-94 from the Department of Electrical Engineering and Computer Science, University of Michigan; ISI/SR-94-379 from the University of Southern California Information Sciences Institute; and CMU-CS-94-134 from Carnegie Mellon University. The best overview of this project was published separately as "Intelligent Agents for Interactive Simulation Environments", by Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S. and Schwamb, K., in AI Magazine, 16(1), 1995.

The research covered in these papers spans a wide spectrum of issues in agent development such as learning [2], planning [3], coordination, and command and control [4,6], natural language processing [5], agent tracking [7, 8, 9] and piloting rotary wing aircraft [10].

The papers are organized by having the overview paper first followed by all of the other papers in alphabetic order by author.

1. Laird, J. E., Johnson, W. L., Jones, R. M., Koss, F., Lehman, J. F., Nielsen, P. E., Rosenbloom, P. S., Rubinoff, R., Schwamb, K. Tambe, M.,Van Dyke, J., van Lent, M. and Wray, R. E.
   Simulated Intelligent Forces for Air: The Soar/IFOR Project 1995. Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation. Orlando, FL. pp. 27-36, May 1995.

2. Johnson, W. L. and Tambe, M.
   Using Machine Learning to Extend Autonomous Agent Capabilities. The Proceedings of the 1995 Summer Computer Simulation Conference, Society of Computer Simulation, 1995.

3. Jones, R. M., Wray, R. E., van Lent, M., and Laird, J. E.
   Planning in the Tactical Air Domain. In Planning and learning: On to real applications, papers from the 1994 AAAI Fall symposium (Technical Report No. FS-94-01). Menlo Park, CA: AAAI Press.

4 Laird, J. E., Jones, R. M. and Nielsen, P. E.
   Multiagent Coordination in Distributed Interactive Battlefield Simulations, Abstract published in Proceedings of the International Conference on Multi-agent systems (ICMAS). June, 1995.

5. Lehman, J. F., Van Dyke, J., and Rubinoff, R.
   Natural Language Processing for IFORs: Comprehension and Generation in the Air Combat Domain. Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation. Orlando, FL. pp. 115-123, May 1995.

6. Nielsen, P.
Intelligent Computer Generated Forces for Command and Control. Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation. Orlando, FL. pp. 211-218, May 1995.

7. Tambe, M.
Recursive Agent and Agent-group Tracking in a Real-time Dynamic Environment. In Proceedings of the International Conference on Multi-agent systems (ICMAS). June, 1995.

8. Tambe, M. and Rosenbloom, P. S.
RESC: An Approach for Real-time, Dynamic Agent Tracking. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), August, 1995.

9. Tambe M., Schwamb, K., and Rosenbloom, P. S.
Building Intelligent Pilots for Simulated Rotary Wing aircraft. In Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation. pp. 39-44, May, 1995.

This technical report is being published concurrently at the University of Michigan (CSE-TR-242-95), the University of Southern California Information Sciences Institue (ISI/SR-95-406), and Carnegie Mellon University (CMU-CS-95-165).

This research was supported under contract N00014-92-K-2015 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Research Laboratory, and contract N66001-95-C-6013 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Command and Ocean Surveillance Center, RDT&E division.

Simulated Intelligent Forces For Air: The Soar/IFOR Project 1995

John E. Laird,[1] W. Lewis Johnson,[2] Randolph M. Jones,[1] Frank Koss,[1] Jill F. Lehman,[3]
Paul E. Nielsen,[1] Paul S. Rosenbloom,[2] Robert Rubinoff,[3] Karl Schwamb,[2]
Milind Tambe,[2] Julie Van Dyke,[3] Michael van Lent,[1] and Robert E. Wray, III[1]

[1]Artificial Intelligence Laboratory
University of Michigan
1101 Beal Ave.
Ann Arbor, MI 48109-2110
laird@umich.edu

[2]Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
rosenbloom@isi.edu

[3]Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
jef@cs.cmu.edu

1 Abstract

For the last three years, the Soar/IFOR group has been developing intelligent forces for distributed interactive simulation environments. Since early 1994, our efforts have been focused on developing computer generated forces for air missions including both fixed wing and rotary wing aircraft. This paper reviews the current state of the Soar/IFOR project and discusses the results of a preliminary trial of our agents in STOW-E, a precursor to STOW-97.

2 Introduction

The goal of the Soar/IFOR project is to develop human-like synthetic agents for populating interactive distributed simulation environments. In contrast to the standard semi-automated forces (SAF) approach, where it is assumed that some higher-level authority, such as a human or a computerized command force (CFOR), will be responsible for all decisions requiring judgement, our approach is to endow all entities with knowledge and decision making abilities similar to those found in humans performing similar tasks. Our hypothesis, confirmed in part by our participation in a large scale simulated exercise called STOW-E, is that building intelligent forces provides a payoff in terms of increasing the fidelity of the agents' behavior, while decreasing the complexity of commanding the agents.

From 1992 through early 1994, our efforts were focussed on research and development for beyond visual range air-to-air combat leading to the creation of TacAir-Soar [Jones *et al.*, 1993; Rosenbloom *et al.*, 1994; Tambe *et al.*, 1995a]. In early 1994, we broadened our horizon significantly, and we are now working on developing automated synthetic pilots for the majority of air missions flown in the U.S. military. The various missions include air-to-air (defensive combat air patrols, sweeps), air-to-ground (close air support, interdiction, strategic attack), air-to-surface, rotary wing (anti-armor), as well as some support missions (refueling, resupply, etc.). We are also developing additional agents, such as air and ground controllers, that communicate with the agents flying in planes and helicopters during their missions. We will refer to all the agents being developed by the Soar/IFOR project as Air-IFOR agents, while TacAir-Soar refers to the agents that fly tactical fixed wing aircraft.

During the last year, we have made progress on many of these missions, and in this paper we will review all aspects of the existing Soar/IFOR agents, including: the interaction between Air-IFOR agents and DIS, the design of Air-IFOR agents, their capabilities, the interactions between multiple Air-IFOR agents, and the participation of Air-IFOR agents in STOW-E.

3 Interaction with DIS

Since the inception of the Soar/IFOR project, our goal has been to create an abstract interface layer between Air-IFOR agents and the underlying simulation (DIS) environment. We call this

the "virtual cockpit" abstraction, meaning that Air-IFOR agents should have an interface that supports the types of interactions a pilot has in the cockpit of a plane or helicopter [Schwamb *et al.*, 1994]. Thus, Air-IFOR agents are isolated from the details of the underlying simulation environment, network protocol, plane dynamics, sensor simulation, etc. Currently, we use ModSAF [Calder *et al.*, 1993] as the underlying software which provides connectivity to the DIS environment as well as simulations of the vehicle dynamics, sensors, weapons, and communication (radio) systems. To support the virtual cockpit, we have added C code, which defines a Soar/ModSAF Interface (SMI)[Schwamb *et al.*, 1994]. The SMI makes all of the appropriate calls to the underlying ModSAF functions so that Air-IFOR agents get access to the appropriate sensor and weapons systems. The SMI does not use ModSAF tasks or taskframes, but instead relies on lower level functions which gives Air-IFOR agents finer-grain control of their own behavior.

Air-IFOR agents are built within the Soar architecture [Laird *et al.*, 1987; Laird and Rosenbloom, 1994; Rosenbloom *et al.*, 1991; Rosenbloom *et al.*, 1993]. Soar, the SMI, and ModSAF are integrated (within the same Unix process) so that each Soar/IFOR agent gets "ticked" during the simulation cycle. Using this arrangement, we can run multiple, independent agents on a single Unix workstation, as well as having agents on many different machines — although a single agent is not distributed across multiple machines. Air-IFOR agents do not share data except through explicit communication using simulated radios.

As part of building the SMI, we have extended the standard suite of ModSAF sensors and weapons, adding such devices as a CCIP (continuously computed impact point) which displays where a bomb will hit if released, a waypoint computer which displays the appropriate heading to fly to the next waypoint in a flight plan, air-to-surface missiles (such as the Exocet), and a primitive form of precision-guided munitions.

One result of our development has been the recognition that the closer we model the types of information available to humans, not at the level of visual perception, but instead at the level of symbolic data, the easier it is to model the behavior of the humans. For example, we discovered that creating a waypoint computer and the CCIP greatly reduced the reasoning required by the Soar agents because they no longer had to respond to every change in their position relative to a waypoint or target. Instead they could respond to the changes in the heading suggested by the waypoint computer or CCIP.

A problem we foresee in the future is the management of many Soar/IFOR agents during a protracted exercise. The problem is not in terms of command and control (covered in the section on multiagent interactions), but is in terms of managing the creation, reuse, and destruction of Air-IFOR agents on many different workstations. To this end, as well as to support cleaner interfaces to Soar agents, we have integrated Soar with Tcl [Ousterhout, 1994], a scripting language, that will help support agent management across many machines.

4 Agent Design

The overall design of Air-IFOR agents has not changed significantly over the last year, although it has been refined and augmented with new tools. Nor have the basic requirements of Air-IFOR agents changed. They continue to be the following:

1. Encode large bodies of knowledge about relevant aspects of the world, including tactics, doctrine, sensors, weapons, etc.
2. React quickly to the environment, such as the behavior of enemy planes, communications from other friendly agents, and changes in terrain being traversed.
3. Determine the tactically relevant features of a complex, dynamic environment.
4. Coordinate behavior with other agents.
5. Use minimal computational resources.
6. Deliberately plan aspects of missions not specified in orders.

4.1 Method and Approach

All of the Soar/IFOR agents are developed within the Soar architecture. Soar has its roots in early AI symbolic systems such as LT [Newell and Simon, 1956], and GPS [Ernst and Newell, 1969], as well as rule-based systems, such as OPS5 [Forgy, 1982]. Soar supports the above requirements by providing two integrated levels of computation: deliberate, sequential operators within problem spaces, and automatic parallel rules. In terms of the tasks that have to be performed by Air-IFOR agents, it is easiest to think in terms of the first level, operators. We make the claim that sequences of deliberate operators are the most appropriate way to model the second to second behavior of a pilot (or any human for that matter). Example operators include flying a mission, picking a control point to fly to, intercepting a bandit, entering a waypoint into the plane's waypoint computer, deciding which missile to fire, physically selecting that missile, pushing the fire button, and so on. Some of these are purely mental operators, such as deciding which missile to select, while others include physical actions. Many

of these operators cannot be performed directly as a single act, but instead must be decomposed into subgoals where finer-grain operators are selected and applied. For example, the act of intercepting a bandit is decomposed into many different operators, such as achieving proximity, employing weapons, and so forth.

Thus, Soar organizes the doctrine and tactics of flying missions in planes and helicopters in terms of hierarchies of operators. For a given operator that the agent is trying to pursue, such as an intercept, the operators used to achieve it are grouped in terms of *problem spaces*. They are called problem spaces because their constituent operators determine the space of problems that can be solved. Operators can be shared among more than one problem space. For example setting the waypoint computer is used in flying routes, as well as flying BARCAPs. Other, so-called, *floating operators*, are available in every active problem space. Floating operators such as operators that detect changes in a bogey's activity, are very sensitive to changes to the environment and usually need to be selected soon after they become relevant. More generally, the hierarchical and floating operators can be seen as at opposite ends of two dimensions: sensitivity to the agent's current goals, and sensitivity to the current situation. All operators must be sensitive to both concerns, but floating operators emphasize reacting to the current situation (within the context of the current goals), while hierarchical operators emphasize responding to the current goals (within the context of the current situation).

Within a subgoal, local situational information is held in the subgoal's *state*. Each subgoal has access to all of the state information in its supergoals, and the state of the top goal contains all the data used to fly a mission, including all sensor data, the agent's interpretation of the current situation, a description of the current mission, data on other agents, etc.

The hierarchical operator structure provides the necessary framework for encoding knowledge and organizing the behavior of Air-IFOR agents; however, it alone is insufficient to provide flexibility and reactivity. What is needed is the ability to dynamically propose, select, and apply the operators that are appropriate for the current situation. This is done in Soar through its underlying rule-base system, which directly implements the selection, application, and termination of operators described above. Thus, there are rules which test the current situation and propose operators, rules which compare proposed operators and suggest *preferences* between operators, rules which test that an operator has been selected and then performs some aspect of the operator, and rules that test that all aspect of an operator have been completed, and signal that the operator is finished. The actual selection of operators is not done directly by individual rules, but by a *decision procedure*, which selects an operator based on all relevant preferences.

Most rule-based systems use a conflict-resolution scheme to select a single rule to fire on each cycle. However, rules from these systems map more directly onto Soar's operators, which are the locus of deliberate activity in Soar, and where selection is controlled by preferences and the decision procedure. Soar's rules are more like an associative memory, where the information in actions of rules is recalled whenever the conditions of the rules match. Thus to retrieve all information relevant to the current situation, the basic cycle is to fire all rules that match the current situation, and continue firing until quiescence. During this rule firing phase, rules to implement the current operator are firing, as well as rules proposing new operators. At quiescence, assuming the current operator is finished, a decision is made to select a new operator based on the available preferences, and the cycle begins again. If the current operator cannot be finished, possibly because it requires problem solving in a subgoal, a subgoal will be created automatically, and then rules sensitive to the subgoal will fire to suggest appropriate operators. When a rule detects that the original operator is finally complete (or should be abandoned), it will fire and cause a new operator to be selected and the immediate subgoal (and any additional subgoals) will be automatically removed. Soar is integrated with ModSAF so that one decision is made for each agent during each clock tick of the simulation, and thus 2 to 15 decisions are made in each Air-IFOR agent each second.

4.2 Infrastructure

In maintaining a rule-based system, the rules must be organized so that it is easy to find rules, not only by their name, but also by their role in producing behavior. For the Soar/IFOR agents, we have mapped the hierarchical structure of the operators onto the hierarchical structure of the Unix file system. Thus, each goal (or subgoal) has its own directory, and within that directory there are files for each of the operators, plus a file for loading in those operator files. For cases where rules are not shared across agents, we have a dynamic load facility that loads only the subset of the code that is relevant to the current agent's vehicle and mission.

Our lowest-level documentation of the problem space, operators, and rules is also organized in the same hierarchical file structure with direct links from the documentation to the code [Koss and Lehman, 1994]. A higher level of documentation,

using the terminology and structure of our domain experts, links into the problem space documentation to currently support a limited form of validation. All of our documentation is in HTML and it can be accessed through viewers such as Mosaic and Netscape.

To support the creation of the code and documentation with our conventions, we have created the Soar Development Environment (SDE) [Hucka and Laird, 1995], which is an extension to Emacs. SDE has a template language that can be used to automatically generate all of the necessary directories, code, and documentation files when new operators are created. SDE also provides many features to aid in debugging, such as automatic finding of files in which rules are stored, point and click commands for common functions, and general search facilities for the rules.

4.3 Current Status and Lessons Learned

The current Air-IFOR agents have a combined total of approximately 320 operators, with a total of 3,100 rules. Individual agents have between 1,130 and 2,550 rules depending on their missions. These counts do not include our natural language or debriefing systems, which by themselves have substantial numbers of rules.

One of the challenges in building the agents has been to maintain the computational efficiency of the system as we add new capabilities. The problem is not that Soar slows down as the sheer number of rules increase (research indicates that Air-IFOR agents may be able to grow to even a million rules without this being an issue [Doorenbos, 1994]), but instead the problem is that it is easy to write rules that fire every time some input data changes (such as when the current position of the plane changes). As a result, we closely monitor rule firings in order to identify costly rules, and then attempt to rewrite them in order to decrease their cost. In a few cases, we have discovered that by removing a computation from Soar that is done in the cockpit for a pilot, such as with the waypoint computer and the CCIP, we have been able to drastically reduce the computational overhead in Soar.

During agent development, we are able to run 6-10 agents on a single 150MHz 4400 SGI Indy. However, one of the lessons we learned from STOW-E is that we are limited to around 4 agents when there are large numbers of entities on the network. This is because of overhead in both ModSAF and the Soar agents that results from the processing of large numbers of entities. In response, we expect to put more emphasis on focusing attention on only the most important entities at all levels of processing, as well as to continue research on efficient matching of rule-based systems [Acharya and Tambe, 1993; Kim and Rosenbloom, 1993].

5 Agent Capabilities

Although Soar provides the basic architecture for building Air-IFOR agents, our agents are more than a large collection of rules that directly encode doctrine and tactics. They must also have a many cognitive capabilities, some of which are directly related to military flying such as following a flight plan, situational awareness, planning attacks, employing weapons, and managing fuel, while others are more general cognitive capabilities, such as communicating with other agents, modeling the behavior of other agents, being able to explain the agent's behavior, and using general problem solving strategies.

To date, we have discovered that although these general cognitive capabilities are important, we have been able to build viable agents by concentrating on those capabilities directly related to performing our agents' missions. Thus, we have developed and incorporated capabilities for following flight plans, planning attacks, employing weapons, situational awareness, managing fuel, and so on. All of these are the building blocks for various missions. There are also many capabilities dealing with coordinating behavior among multiple agents, which are discussed in the section on multiagent interactions. These capabilities are all implemented as operators that have complex subgoals. For example, following a flight plan involves many operators including flying routes (of which there are different types depending on the aircraft), performing various activities at waypoints (such as communicating with control agents or determining if a plane should delay at the point so that it arrives on target at the appropriate time), selecting the next route, and processing any changes the agent might receive to its mission. We expect these capabilities to be reused on future missions, possibly with modification as new variants are required.

We expect that the more general cognitive capabilities will become necessary as we try to create agents which are more autonomous, and thus able to handle novel situations on their own. To that end, we are pursuing research in the following areas:

1. Natural language processing: Even with the advent of the Command and Control Simulation Interface Language (CCSIL) [Salisbury, 1995], we will someday want Air-IFOR agents to directly interact with humans. Air-IFOR agents will need to understand and generate natural language, with one of the challenges being to integrate the processing of language

with all of the other agents' tasks [Lehman *et al.*, 1995].

2. Behavior explanation: As the complexity of Air-IFOR agents grow, it is necessary for each of them to be able to explain its own behavior and internal reasoning. What action did it take, why did it take that action, why did it interpret the situation in the way it did, and what were other options? We have been actively pursuing these issues in the Debrief system, which is a set of Soar rules that when included in an agent before a run, allows the agent to be debriefed after flying a mission [Johnson, 1994].
3. Agent modeling: In order to interpret the actions of other agents, Air-IFOR agents must have some understanding of what the other agents are thinking. This is currently done in very specialized and context specific ways in Air-IFOR agents. However, as we start to explore complex behavior, it will be necessary for Air-IFOR agents to create general internal models of what other agents are thinking about the current situation. For example, deceptive maneuvers involve generating behaviors with the goal of leading an opponent to incorrectly guess what your intent and action really is. We can currently encode "deceptive" maneuvers in Air-IFOR agents; however, for the agent itself to derive an appropriate deceptive maneuver in novel situations requires the ability to internally model some of the thought processes of other agents, a problem we are actively pursuing [Tambe and Rosenbloom, 1995].
4. General Problem Solving and Planning: Our current agents have all the necessary control knowledge for making the decisions we expect them to encounter. However acquiring this knowledge is difficult and time-consuming, and this knowledge alone does not always lead to robust performance in novel situations. Over the last year, we have done research on more general problem solving and planning approaches that can use more "fundamental" knowledge of the domain and thus increase the ability of Air-IFOR agents to respond to novel situations. Using experimental versions of TacAir-Soar, we have demonstrated the feasibility of integrating both look-ahead planning [van Lent, 1995] and means-ends analysis [Wray, 1995] into Air-IFOR agents.

In addition to the more general capabilities listed above, Air-IFOR agent must have knowledge that includes the doctrine and tactics appropriate to the missions they are to perform. Currently, Air-IFOR agents fly the following fixed-wing missions: BARCAP, Close Air Support, Strategic Attack, and MiGSweep. For rotary wing, Air-IFOR agents can fly a basic anti-armor mission [Tambe *et al.*, 1995b]. In addition, we have developed the following agents that act as controllers during missions [Nielsen, 1995].

- Air Intercept Controller (AIC) and Ground Controlled Intercept (CGI) which give information and commands about enemy planes. The AIC is situated in a plane with a large radar, such as an E2C.
- Forward Air Controller (FAC) which provides final directions for close-air support missions.
- Direct Air Support Center (DASC) assigns aircraft to missions, can change the mission, and hands off control to the FAC.
- Fire Support Coordination Center (FSCC) determines the type of support to utilize (close air support, artillery, or naval gunfire) and if close air support is determined it generates a tactical air request form then sends the request to the DASC.
- Tactical Air Command Center (TACC) which provides air traffic control, intermediate routing, and deconfliction.
- Tactical Air Direction (TAD) controller directs specific air operations within the area of operations, prior to the establishment of a DASC.

We have operational versions of all of these agents, although many are limited to producing behavior that is only relevant to close air support and air-to-air missions.

6 Multiagent Interactions

Although the individual agents are by themselves important, it is the coordination of agents that leads to effective military forces. Our approach is to model the methods and practices of military organizations. Air-IFOR agents coordinate their activities through a combination of common background knowledge (their knowledge of military methods, procedures, doctrine and tactics), common mission statements, and explicit communication (non-verbal and verbal) [Laird *et al.*, 1995]. Because Air-IFOR agents know what they are supposed to do and when (because of their background knowledge and mission statements), the need for explicit communication is greatly reduced. Also, in contrast to SAF agents, Air-IFOR agents are "smart" enough to deal with the details of executing all aspects of the missions they have been assigned and do not require constant monitoring by a human or command agent. When explicit verbal communication is used, we attempt to model both the content and form used by real pilots. Thus, Air-IFOR agents send simulated radio messages whose content closely mirrors the English words

and phrases used by real pilots. The generation and interpretation of these messages is currently done by a fixed set of templates and not a general-purpose natural language facility (although one is under development [Lehman *et al.*, 1995]). Air-IFOR agents currently can generate and interpret approximately 100 different types of messages.

When flying as a unit, most of the coordination occurs by the wingman visually observing and responding to the behavior of the lead of the unit. The wingman constantly adjusts its position to stay in the appropriate formation. The wingman also keeps track of the progress of the unit in its mission, observing the achievement of waypoints. Depending on the mission details, the wingman may change formation, break formation to fly an independent ground attack, rejoin the formation following an attack, or even take over as the lead.

Currently, TacAir-Soar agents (Air-IFOR agents for tactical fixed wing aircraft) are able to fly as either sections (two planes) or divisions (four planes). They can fly a variety of formations and they can dynamically break into smaller units, such as a division splitting into two sections, and then later reform as a single unit. Within a section, the lead and wingman can coordinate their radars (covering different parts of the sky and communicating enemy contacts) as well as coordinating their weapons employment during air-to-air engagments. During air-to-ground attacks, a section can use a variety of coordinated tactics, which are planned by the lead at the beginning of the mission. Our work on coordination with rotary wing units is also under development where currently the helicopters can fly in pairs, with the expected progression to platoons and then companies during the next year.

A unit of TacAir-Soar agents, such as a section or division, will also coordinate its behavior with available controllers (AIC, CGI, FAC, TAD, TACC, FSCC, DASC) [Nielsen, 1995]. The controllers can give the unit flight information (such as the altitude to fly at, or the name of the next controller), permission to continue the mission (permission to enter an area, or permission to attack a target), information on other planes, or changes to missions. In the case of changing a mission, a controller can dynamically change almost any aspect of a ground attack mission including the route, the time on target, and the final target. When a mission change is received, the members of the unit change their missions, sometimes replanning the final attack for air-to-ground missions.

Our goal is to continue to build up the coordination of Air-IFOR agents into integrated missions. We are currently close to completing close-air support which involves a variety of controllers plus planes doing individual missions. However, missions such as offensive strike and integrated interdiction can involve a variety of different planes flying many different individual missions (strategic attack, RECCE, MiGSweep, SEAD, etc.) that have to be closely orchestrated to pull off the complete mission. We plan on working on these missions and the required coordination over the next year.

Our approach to date has been to support the coordination of activities within the set of agents under our direct control. We have been able to develop our own templates independent of other groups. However, in the future some Air-IFOR agents will need to communicate with other command forces, and thus, we will soon be using CCSIL protocols for communication between our agents and their controllers.

7 STOW-E

During November 4–7, 1994, a large scale operational military exercise called STOW-E (Simulated Theater Of War – Europe) was held across 18 installations in United States and Europe. At its peak, over 1,800 entities were simulated on the Defense Simulation Internet (DSI). Although the vast majority of the entities were involved in ground actions, there were also a significant number of air missions being flown using humans in simulators, ModSAF agents, Soar/IFOR agents, and in a few cases, real planes with instrumentation that allowed them to be sensed within the DIS environment (although these planes could not sense the DIS entities). For the Soar/IFOR group, this was the first chance to participate in a realistic, large scale simulation environment where we did not have complete control over the scenarios.

Over the four day period, the Soar/IFOR agents were scheduled to participate in 10 events. For each event we had specific missions assigned to Air-IFOR agents that had been given to us weeks in advance. These missions included defensive air missions (BARCAPs), offensive air missions to disrupt BARCAPs, air to ground missions, and air to surface missions.

We successfully fielded agents for every event in which we were scheduled (10 events, approximately 32 agents) and participated in many unscheduled events (5–7 events, approximately 16 agents). TacAir-Soar performed air-to-air missions against ModSAF and humans (in simulators). TacAir-Soar attempted to engage planes from other sites, but because of problems with the network, the other agents did not see TacAir-Soar. We also participated in air-to-ground (bombing bridges, etc.) and air-to-surface (firing missiles at ships) attacks in which we engaged ground and surface targets from other sites.

We did have a limited number of software failures with the most significant being our inability to fly over the terrain database where the ground battle was raging when it was populated with hundreds of tanks. This was caused by a software bug in our C code for processing ground targets using radar.

One of our goals was to provide viable opponents for simulated and human pilots; however it was difficult to evaluate the "skill" of our TacAir-Soar because of some problems with the underlying simulation models. For example, during the first day, we were frustrated with the performance of TacAir-Soar in engagements. They were easily shot down by ModSAF F/A-18's. We later learned that in order to populate the simulation with different types of planes, the F/A-18's were created by copying F-14's. The F/A-18's were therefore carrying Phoenix missiles which are much longer range than any missile carried by an F/A-18. TacAir-Soar, basing its tactical behavior on the known properties of F/A-18's, was caught by surprise (as it should have been).

In engagements with humans, our planes would often get into good tactical positions, only to see our missiles miss when they were shot. (TacAir-Soar did have some kills against humans in simulators, but in general, TacAir-Soar got "toasted".) We believe that the missile missed because of flaws in the ModSAF missile models. Thus, although TacAir-Soar got shot down, it was in general using appropriate tactical maneuvers. Independent of the specific outcome, this exercise proved the value of taking systems out of the laboratory and testing them in more realistic situations.

Possibly the best example of our capabilities was in the execution of an unscheduled event for the second day. In this mission, a section of F/A-18's were to perform a ground attack against a set of islands in the simulated battle area. Our planes were used in place of a virtual (manned) ground attack because of the failure of that simulator. Enroute to the target, the planes were unexpectedly intercepted by ModSAF MiG-29's. The F/A-18's engaged the MiG-29's to defend themselves and got off one or two shots (but no kills). The MiG-29's disappeared from the network, and our planes automatically returned to their air-to-ground attack mission. Further enroute, they were unexpectedly fired on from a surface-to-air site, killing the wingman (not only did the planes not expect it, we didn't realize there would be any surface-to-air systems in STOW-E — clearly an unscripted interaction). The lead continued on, successfully dropping bombs on the designated target and then egressing back to base.

Although we considered our participation in this exercise a success, it did demonstrated some weaknesses that we must address in the future.

- Number of vehicles: We discovered that for an exercise with a large number of vehicles, we were not able to run the number of vehicles/workstation that we had expected. Part of this is the overhead in the network processing code of ModSAF, but it also was a problem for our AIC/E2C agent which could see a large number of agents at once because of its radar. This has led us to use more deliberate focusing of attention in Air-IFOR agents so that they do not attempt to process the complete situation at once, but instead concentrate on subsets of the situation, preferably those that are relevant to the current tactical situation.
- Mission set up: Before STOW-E, we had not developed any tools to help specify and manage the missions of Air-IFOR agents. During STOW-E, it was time-consuming and error-prone for us to create or modify the missions. As a result, we are currently developing graphical interface tools that will make it possible to enter and modify missions directly, without editing intermediate data structures. Our goal is that our interface should give the user the same look and feel as the documents and tools used by pilots in their normal briefings. The integration of Tcl and Soar is making this much easier because of its ability to manage windows and build formated graphical and textual interfaces. In the future we must also have the ability to accept missions from other software systems using CCSIL; however the details of the protocols have yet to be defined.
- Runtime control: Once Air-IFOR agents received their missions, they would fly the missions without any human management. Thus, we became observers and ran our exercises "hands-off". In contrast, the ModSAF planes required constant attention, with a human controlling their behavior on and off during the exercise. Although we wish to continue our approach, we also came to recognize that we needed the ability to dynamically change some aspects of the missions of Air-IFOR agents during the exercise, such as changing the waypoint at which a section of planes is stationed. These are relatively minor changes to TacAir-Soar.

This exercise has the additional significance of demonstrating that "hard core" AI technology can be successfully used in an operational exercise (although in STOW-E this was in a limited role). We believe that this is one of the first (if not the first) time that an AI system has been used in this way.

8 Summary and Conclusions

In the beginning of the Soar/IFOR project, there were many questions as to whether it was practical to develop intelligent forces for synthetic environments. Although there is still much more work to do, three years of research and development have brought us to the point where we can state with some degree of certainty that intelligent forces are practical and will play a significant role in STOW-97. It is difficult to isolate specific parts of our methodology or underlying technology as responsible for this success, although clearly we believe that the underlying Soar architecture is responsible to a significant degree. Its ability to combine fine-grain reactive reasoning of rules, with more deliberate and hierarchical decision making using operators within problem spaces, appears to be well matched to the demands of the interactive simulation and the cognitive processes of the humans we are attempting to model.

One surprise has been our ability to build complex and relatively general systems while not using many of the more advanced techniques such as means-ends analysis, planning, learning, complex agent modeling, or natural language. However, we still believe that these are critical capabilities for building robust, general agents, and we are continuing to pursue research in these areas.

In the immediate future, we will continue to expand the breadth of missions and capabilities of Air-IFOR agents. For fixed wing, a primary goal is to develop the appropriate agents to fly integrated interdiction and strategic attack missions. The coordination of many different types of aircraft, with different missions promises to be challenging. In rotary wing, our goal is to field a complete company of attack helicopters. Our plan is for these developments to lead up to a successful participation of Soar/IFOR agents in STOW-97.

9 Acknowledgments

This research was supported under contract N00014-92-K-2015 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Research Laboratory, and contract N66001-95-C-6013 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Command and Ocean Surveillance Center, RDT&E division. The authors would like to thank BMH Associates, Inc. for their technical assistance.

References

[Acharya and Tambe, 1993] A. Acharya and M. Tambe. Collection-oriented match. In *Proceedings of the Second International Conference on Information and Knowledge Management*, November 1993.

[Calder *et al.*, 1993] R. Calder, J. Smith, A. Courtenmanche, J. Mar, and A. Ceranowicz. ModSAF behavior simulation and control. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, 1993.

[Doorenbos, 1994] R.B. Doorenbos. Combining left and right unlinking for matching a large number of learned rules. In *Proceedings of AAAI-94*, Seattle, WA, August 1994.

[Ernst and Newell, 1969] G. W. Ernst and A. Newell. *GPS: A Case Study in Generality and Problem Solving.* Academic Press, New York, 1969.

[Forgy, 1982] C. L. Forgy. Rete: A fast algorithm for the many pattern / many object pattern match problem. *Artificial Intelligence*, 19:17–38, 1982.

[Hucka and Laird, 1995] M. Hucka and J. E. Laird. The Soar Development Environment. Technical report, The University of Michigan, Department of Electrical Engineering and Computer Science, 1995.

[Johnson, 1994] W.L. Johnson. Agents that learn to explain themselves. In *Proceedings of AAAI-94*, pages 1257–1263, Seattle, WA, August 1994. AAAI, AAAI Press.

[Jones *et al.*, 1993] R. M. Jones, M. Tambe, J. E. Laird, and P. S. Rosenbloom. Intelligent automated agents for flight training simulators. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pages 33–42, Orlando, FL, 1993.

[Kim and Rosenbloom, 1993] J. Kim and P. S. Rosenbloom. Constraining learning with search control. In *Machine Learning: Proceedings of the Tenth International Conference*, pages 174–181, San Mateo, CA, 1993. Morgan Kaufmann.

[Koss and Lehman, 1994] F. V. Koss and J. F. Lehman. Knowledge acquisi-

tion and knowledge use in a distributed IFOR project. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 1994.

[Laird and Rosenbloom, 1994] J. E. Laird and P. S. Rosenbloom. The evolution of the Soar cognitive architecture. Technical report, Computer Science and Engineering, University of Michigan, 1994. To appear in Mind Matters, T. Mitchell Editor, 1995.

[Laird *et al.*, 1987] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(3), 1987.

[Laird *et al.*, 1995] J. E. Laird, R. M. Jones, and P. E. Nielsen. Multiagent coordination in distributed interactive battlefield simulations. Technical report, Computer Science and Engineering, University of Michigan, 1995.

[Lehman *et al.*, 1995] J. F. Lehman, J. Van Dyke, and R. Rubinoff. Natural language processing for IFORs: Comprehension and generation in the air combat domain. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 1995.

[Newell and Simon, 1956] A. Newell and H. A Simon. The logic theory machine: A complex information processing system. *IRE Transactions on Information Theory*, IT-2:61–79, September 1956.

[Nielsen, 1995] P. Nielsen. Intelligent computer generated forces for command and control. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 1995.

[Ousterhout, 1994] J. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, MA, 1994.

[Rosenbloom *et al.*, 1991] P. S. Rosenbloom, J. E. Laird, A. Newell, and R. McCarl. A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47, 1991.

[Rosenbloom *et al.*, 1993] P. S. Rosenbloom, J. E. Laird, and A. Newell. *The Soar Papers: Research on Integrated Intelligence.* MIT Press, 1993.

[Rosenbloom *et al.*, 1994] P. S. Rosenbloom, W. L. Johnson, R. M. Jones, F. Koss, J. E. Laird, J. F. Lehman, R. Rubinoff, K. B. Schamb, and M. Tambe. Intelligent automated agents for tactical air simulation: A progress report. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, May 1994.

[Salisbury, 1995] M. Salisbury. Command and Control Simulation Interface Language (CCSIL): Status update. In *Proceedings of the the 12th Distributed Interactive Simulation Workshop*, 1995. Sponsored by STRICOM and the Institute for Simulation and Training (IST) at the University of Central Florida.

[Schwamb *et al.*, 1994] K. B. Schwamb, F. V. Koss, and D. Keirsey. Working with ModSAF: Interfaces for programs and users. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, May 1994.

[Tambe and Rosenbloom, 1995] M. Tambe and P. S. Rosenbloom. Agent tracking in complex multi-agent environments: New results. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 1995.

[Tambe *et al.*, 1995a] M. Tambe, W. L. Johnson, R. M. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), 1995.

[Tambe *et al.*, 1995b] M. Tambe, K. Schwamb, and P. S. Rosenbloom. Building intelligent pilots for simulated rotary wing aircraft. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 1995.

[van Lent, 1995] M. van Lent. Planning and learning in a complex domain. Technical report, The University of Michigan, Department of Electrical Engineering and Computer Science, 1995.

[Wray, 1995] R. E. Wray. A general framework for means-ends analysis. Technical report, The University of Michigan, Department of Electrical Engineering and Computer Science, 1995.

11 Biographies

John E. Laird is an associate professor of Electrical Engineering and Computer Science and the director of the Artificial Intelligence Laboratory at the University of Michigan. He received his B.S. degree in Computer and Communication Sciences from the University of Michigan in 1975 and his M.S. and Ph.D. degrees in Computer Science from Carnegie Mellon University in 1978 and 1983, respectively. His interests are centered on creating integrated intelligent agents (using the Soar architecture), leading to research in problem solving, complex behavior representation, machine learning, cognitive modeling.

W. Lewis Johnson is a project leader at the University of Southern California Information Sciences Institute, and a research assistant professor in the USC Department of Computer Science. Dr. Johnson received his A.B. degree in Linguistics in 1978 from Princeton University, and his M.Phil. and Ph.D. degrees in Computer Science from Yale University in 1980 and 1985, respectively. He is interested in applying artificial intelligence techniques in the areas of computer-based training and software engineering. Dr. Johnson is co-editor-in-chief of the journal Automated Software Engineering, Secretary/Treasurer of the SIGART Bulletin, and is on the steering committee of the AI and Education Society.

Frank V. Koss is a systems research programmer in the Artificial Intelligence Laboratory at the University of Michigan, where he is developing the interface between the Soar cognitive architecture and the ModSAF simulator and extending ModSAF itself. He received his BS in computer engineering from Carnegie Mellon University in 1991 and his MSE in computer science and engineering from the University of Michigan in 1993.

Jill Fain Lehman is a research computer scientist in Carnegie Mellon's School of Computer Science. She received her B.S. from Yale in 1981, and her M.S. and Ph.D. from Carnegie Mellon in 1987 and 1989, respectively. Her research interests span the area of natural language processing: comprehension and generation, models of linguistic performance, and machine learning techniques for language acquisition. Her main project is NL-Soar, the natural language effort within the Soar project.

Randolph M. Jones received his Ph.D. in Information and Computer Science from the University of California, Irvine, in 1989. He is currently an assistant research scientist in the Artificial Intelligence Laboratory at the University of Michigan. His research interests lie in the areas of intelligent agents, problem solving, machine learning, and psychological modeling.

Paul E. Nielsen is an assistant research scientist at the Artificial Intelligence Laboratory of the University of Michigan. He received his Ph.D. from the University of Illinois in 1988. Prior to joining the University of Michigan he worked at the GE Corporate Research and Development Center. His research interests include intelligent agent modeling, qualitative physics, machine learning, and time constrained reasoning.

Paul S. Rosenbloom is an associate professor of Computer Science at the University of Southern California and the acting deputy director of the Intelligent Systems Division at the Information Sciences Institute. He received his B.S. degree in mathematical sciences from Stanford University in 1976 and his M.S. and Ph.D. degrees in computer science from Carnegie-Mellon University in 1978 and 1983, respectively. His research centers on integrated intelligent systems (in particular, Soar), but also covers other areas such as machine learning, production systems, planning, and cognitive modeling. He is a Councilor and Fellow of the AAAI and a past Chair of ACM SIGART.

Robert Rubinoff is a postdoctoral research fellow in Carnegie Mellon's School of Computer Science. He received his B.A., M.S.E., and Ph.D. from the University of Pennsylvania in 1982, 1986, and 1992, respectively; his dissertation research was on "Negotiation, Feedback, and Perspective within Natural Language Generation". His research interests include natural language processing, knowledge representation, and reasoning. He is currently working on natural language generation within the Soar project.

Karl B. Schwamb is a Programmer Analyst on the Soar Intelligent FORces project at the University of Southern California's Information Sciences Institute. He contributes to the maintenance of the Soar/ModSAF interface software and the Tcl/Tk interface to Soar. He received his M.S. in Computer Science from George Washington University.

Milind Tambe is a research computer scientist at the Information Sciences Institute, University of Southern California (USC) and a research assistant professor with the Computer Sci-

ence Department at USC. He completed his undergraduate education in computer science from the Birla Institute of Technology and Science, India in 1986. He received his Ph.D. in computer science from Carnegie Mellon University in 1991. His interests are in the areas of integrated AI systems, agent modeling, plan recognition, and efficiency and scalability of AI programs, especially rule-based systems.

Julie Van Dyke is a research programmer at Carnegie Mellon University, working on language comprehension in NL-Soar. She is also working toward an MS in Computational Linguistics with a focus on modeling language acquisition.

Michael van Lent is currently a doctoral candidate in the Artificial Intelligence Laboratory at the University of Michigan. He received his B.A. with honors in computer science from Williams College in 1991 and a Master of Science in Computer Science from the University of Tennessee, Knoxville in 1993. Mr. van Lent also worked for the Naval Center for Applied Research in Artificial Intelligence during the summers of 1992 and 1993.

Robert Wray is currently a candidate for the Ph.D degree in computer science at the University of Michigan. He received a Bachelor of Science in Electrical Engineering from Memphis State University in 1988 and a Master of Science in Electrical Engineering from the University of Massachusetts, Dartmouth in 1993. He also worked for the Naval Undersea Warfare Center from 1989 to 1993 as an electronics engineer. His current research, projects and interests in computer science include: intelligent agent architectures, extending traditional artificial intelligence planning paradigms, machine learning, and software engineering.

USING MACHINE LEARNING TO EXTEND AUTONOMOUS AGENT CAPABILITIES

W. Lewis Johnson and Milind Tambe
USC / Information Sciences Institute & Computer Science Dept.
4676 Admiralty Way, Marina del Rey, CA 90292-6695
WWW: http://www.isi.edu/soar/{johnson, tambe}
{johnson,tambe}@isi.edu

To appear in the Procedings of the 1995 Summer Computer Simulation Conference.

Keywords: machine learning, interactive simulation, military

1 Introduction

The Soar/IFOR project is developing human-like, intelligent agents that can interact with humans, and with each other, in battlefield simulations [10]. Our agents play a variety of roles such as fighter pilots, helicopter pilots, and airspace controllers. The fighter pilot agents in particular have been successfully deployed in large-scale simulation exercises, such as the Synthetic Theater of War (STOW) exercise in November, 1994, which modeled a four day battle scenario involving approximately 2000 military vehicles. Autonomous agents such as Soar/IFOR agents are expected to continue to play a major role in battlefield simulations, which in turn are expected to provide an essential tool for military planning and training in the future.

Soar/IFOR agents are implemented in Soar, a problem solving architecture that integrates a number of human cognitive functions, including problem solving, perception, and learning [4]. Learning occurs through the application of a general mechanism called *chunking* that summarizes the results of processing on subgoals, in the form of rules that can apply to similar subgoals in the future. This chunking process is a form of explanation-based learning EBL [7, 6]. Chunking can lead to speedup in learner performance, and is instrumental to the learning of new concepts. Some Soar systems have managed to learn thousands, and even hundreds of thousands, of chunks[2].

From the previous experience with learning in Soar, it was taken as a given that the Soar/IFOR agents could be made capable of applying chunking in service of their performance requirements. The first research question that we focus on in this paper is then the following: What kinds of knowledge can Soar/IFOR agents learn in the combat simulation environment? In our investigations so far, we have found a number of learning opportunities in our systems, which yield several types of learned rules. For example, some rules speed up the agents' decision making, while other rules reorganize the agent's tactical knowledge for the purpose of on-line explanation generation.

Yet, it is also important to ask a second question: Can machine learning make a significant difference in Soar/IFOR agent performance? The main issue here is that battlefield simulations are a real-world application of AI technology. The threshold which machine learning must surpass in order to be useful in this environment is therefore quite high. It is not sufficient to show that machine learning is applicable "in principle" via small-scale demonstrations; we must also demonstrate that learning provides significant benefits that outweigh any hidden costs.

Thus, the overall objective of this work is to determine how machine learning can provide practical benefits to real-world applications of artificial intelligence. Our results so far have identified instances where machine learning succeeds in meeting these various requirements, and therefore can be an important resource in agent development. We have conducted extensive learning experiments in the laboratory, and have conducted demonstrations employing agents that learn; to date, however, learning has not yet been employed in large-scale exercises. The role of machine learning in Soar/IFOR is expected to broaden as practical impediments to learning are resolved, and the capabilities that agents are expected to exhibit are broadened.

2 The Problem Domain

Soar/IFOR agents are designed to work within distributed interactive simulations (DIS) of military exercises. But unlike conventional "semi-automated" entities in distributed simulations, Soar/IFOR agents are fully capable of autonomous decision making without outside human intervention. They are intended to be realistic models of military agent behavior, so much so that to an outside observer their behavior is indistinguishable from that of people. They must perform most if not all of the functions that human personnel would be called upon to perform, e.g., to issue and/or understand commands, to coordinate their activities with friendly forces, and to interpret and respond to the actions of enemy units. Needless to say, achieving these goals successfully is a significant achievement for artificial intelligence.

Soar/IFOR agents interact with distributed simulations via the ModSAF simulation package [1]. Each agent is assigned to a ModSAF simulation of a vehicle, e.g., an aircraft. Soar/IFOR receives inputs from the vehicle, via an abstract interface [8], information similar to what a human controlling the same vehicle in the real world would receive, such as position of the vehicle, presence of enemy vehicles in the area, etc. The Soar/IFOR agent interprets the situation based upon the information received, decides on actions to take, and communicates these to ModSAF as commands for the vehicle to execute. Some of the details of psychomotor control and resource contention are omitted from the model, e.g., a Soar/IFOR pilot controls its aircraft by specifying desired altitudes and headings instead of by simulating stick movements. However, these abstractions do not simplify the agents' decision making task.

Soar/IFOR has been tested in simulated exercises incorporating manned simulation devices such as flight simulators, semi-automated forces, as well as automated forces. Soar/IFOR agents are assigned missions prior to the engagement, and are otherwise left to carry out their missions themselves. Agents are evaluated according to how appropriately they perform in each individual engagement.

Although such exercises are useful for demonstrating agent capabilities, they do not in themselves ensure that Soar/IFOR agents meet the needs of potential users of distributed simulations. For example, in order for users to be certain that agent decision making is realistic, they need to understand the rationales for the agent's decisions. This has led to the development of an automated explanation capability, called Debrief, that enables users to engage agents in a question-answer dialog, in a manner analogous to an after-action review [3].

3 Learning in Soar Agents

The air-combat simulation environment—by virtue of its complex, real-world characteristics—presents Soar/IFOR agents with a number of challenging functional and performance requirements. There are also many ways in which machine learning can help the agents meet these requirements. Chunking in IFOR has been found so far to enable the following functional capabilities and performance improvements.

- Decision making speeds up over time.
- A memory of past episodes is maintained.
- Problem solving knowledge is reorganized in order to support explanation and efficient execution.
- Interpretation of situations and events improves in quality with experience.

A Soar/IFOR agent engages in some of this learning *on-line*, i.e., while it is engaged in simulated combat. Prime candidates for such on-line learning include chunking for speedup, episodic memory and knowledge compilation. However, not all learning can or should occur on line. In particular, some of the learning requires that a Soar/IFOR agent consider the consequences of its decisions, explore alternative decisions, and learn from the results. Because of the real-time pressures of air-to-air combat, a Soar/IFOR agent may not have the free time to engage in such deliberation. Time pressures are certainly not continuous: there can be momentary lulls in activity that could be used for deliberation and learning, but as yet are not. Instead, Soar/IFOR agents rely upon *off-line* analysis for such learning. It waits for the combat situation to terminate, so it can analyze past situations without interruption. This enables the agents to explain their reasoning during after-action review, for example.

Learned chunks are applied to future decisions in the following ways. A chunk learned during an engagement may apply later on within the same engagement. It may apply during after-action review of the engagement. Finally, chunks created during a mission or during after-action review are saved so that they can be employed by agents in future missions and review sessions, enabling the agents to learn from accumulated experience.

3.1 Speeding Up Decisions

In much machine learning research, such as [5], speedup is measured by comparing problem solving time after learning to problem solving time without learning. Such a measure is inappropriate for learning in Soar/IFOR, because chunking does not yield an overall speedup, i.e., it does not reduce the overall duration of the engagement. In other domains such lack of speedup might be attributable to the high cost of matching and retrieving the learned chunks[11]. However, for Soar/IFOR agents, the cost of matching and retrieving learned rules is not much of an overhead. Rather a combination of the following two effects are at work. First, combat simulation involves performing (simulated) physical actions and responding to external events. Learning cannot affect the duration of such actions and events; at best it can reduce the time required to decide on an action or interpret an event. Second, cognitive activity is concentrated in isolated episodes, separated by periods of relative inactivity. Speedups in deliberation contribute very little to reductions in the overall duration of a scenario. For instance, suppose a Soar/IFOR agent decides to launch a missile at an opponent. To that end, it must decide which type of missile to employ, and how best to approach the opponent's aircraft. These decisions take up at most a few seconds. The agent then has to wait, sometimes for up to a minute or more while the opponent gets into its missile firing range. Decision time thus has little or no effect on overall time to intercept the opponent.

Although learning has little effect on the overall duration of engagements, it can make a substantial difference in time-critical situations. In such situations, small delays in an agent's action can jeopardize its survival, or prevent the agent from exploiting momentary advantages over an opponent. For instance, when a Soar/IFOR agent fires a missile at its opponent, the opponent may engage in a missile evasion tactic that can cause it to break radar contact (disappear from the Soar/IFOR agent's radar). The opponent may then turn quickly to fire a missile at the Soar/IFOR agent. This is an extremely time-critical situation. When the opponent turns back after its missile evasion maneuver, the Soar/IFOR agent obtains a new contact (blip) on its radar. This blip could be the opponent, or perhaps a friendly aircraft who has just arrived in radar range. The Soar/IFOR agent must quickly determine the contact's identity, and then launch a second missile before the opponent fires her missile. If the Soar/IFOR agent is delayed in re-establishing the opponent's identity, it may get shot down. Chunking can enable Soar/IFOR agents to arrive at important decisions more rapidly the next time a similar situation is encountered. The end result is that the agents can survive longer, and fight better.

A possible way of measuring speedup might be to measure an agent's reaction time, i.e., the from an external event until the agent's response to that event. This presupposes, however, that the stimuli are controlled so that there is a clear relationship between stimulus and response. However, battlefield engagements are not like controlled laboratory experiments: instead, agents are constantly exposed to a variety of stimuli, and perform a variety of tasks, often at the same time. Reducing the amount of time required to interpret one stimulus often has the indirect effect of enabling the agent to attend to other stimuli that were previously overlooked, such as a second opponent that has just arrived in radar range. This clearly can have an impact on overall agent performance, but in a way that is difficult to quantify.

3.2 Maintaining an Episodic Memory

It is useful for Soar/IFOR agents to have an episodic memory, so that they can recall episodes from previous engagements during after-action review or subsequent missions. Episodic memory can be regarded as an aspect of learning, insofar as the problem solver's reasoning after memory formation is different from that before memory formation. It is instrumental to other types of learning: for example, if an agent can recognize that the current situation is similar to previous situations, it can then apply its previous experience to the new situation.

We have found that chunking can be readily employed to address part of the episodic memory problem, namely to learn to recall the circumstances in which a given event occurred. That is, when presented with a description of an event, chunks fire which recreate a description of the world state that prevailed at that time. Other aspects of episodic memory, such as recalling what events occurred as part of a given mission, are not as yet handled via chunking; the agent instead simply records the events that occur in a conventional list data structure.

The episodic memory mechanism relies on two sets of chunks. The first set consists of *recognition chunks*, which are common in a range of Soar systems. Recognition chunks fire in response to some description that serves as a memory probe, indicating that an instance

matching the probe has been seen before. In the Soar/IFOR case, the memory probe consists of a description of an event, together with a possible state change. If the state change occurred at the time the event was observed, the recognition chunk will fire. These recognition chunks are created in a special episodic-memory subgoal, which is processed whenever the agent notices a significant state change. The second set of chunks are *recall chunks*, which recall the complete state in which an event occurred, when presented with an event description as a memory probe. The first time Soar/IFOR attempts to recall the state associated with an event, it first tries to find an earlier event for which it can recall a state. It then tries to recall which state changes occurred between the earlier state and the state of interest. The previously created recognition chunks identify the relevant state changes. Once the recall process is complete, a recall chunk is created, so that the next time the event is used as a memory probe the state is immediately recalled.

Episodic memory illustrates how chunking can serve as an underlying mechanism for a variety of types of learning besides simple speedup. Such learning may require problem spaces that are specially designed to generate particular types of chunks such as recognition chunks or recall chunks.

3.3 Reorganizing Knowledge

Chunking also enables Soar/IFOR agents to reorganize their knowledge. In knowledge based systems generally, the form in which knowledge is encoded depends upon how the knowledge engineer intends the knowledge to be used. Learning enables knowledge encoded for one purpose, i.e., controlling the agent's behavior, to be employed for other purposes, e.g., explaining the agent's decisions.

Soar/IFOR's interactive explanation capability, called Debrief, makes extensive use of chunking for knowledge reorganization [3]. The agents can explain the rationales for decisions made during an engagement, by relating chosen decisions to the critical factors in the situation that led to those decisions. The knowledge needed to generate such explanations, i.e., associations between decisions and sets of situational factors, is different from the knowledge used to generate the decisions in the first place. For one thing, the process of generating the decision may involve internal reasoning mechanisms that are of little interest to someone who is not an agent developer. Recognition chunks are built which identify the key factors leading to a decision in a given situation. This is accomplished by reconsidering the decisions after the engagement is over, and proposing hypothetical changes to the situation in which the decision was made. The set of state features that prove significant, because altering them alters the outcome of the decision, is saved in a chunk. If the agent is asked to explain a similar decision in a similar situation, the recognition chunk will fire identifying those features of the situation that should be included in the explanation.

Knowledge reorganization also allows knowledge organized for ease of knowledge engineering to be rendered in a form suitable for efficient execution. The Soar/IFOR project is developing a variety of types of agents, among which only some knowledge is shared. Rules therefore tend to be factored so as to separate the shared knowledge from the unshared knowledge. Chunking is used in some cases to combine this knowledge into larger agent-specific rules, thus reducing the number of rules that must execute. This happens because chunking summarizes the results of all rules that are executed in a subgoal, in the form of a single rule that represents their combined effect. Agent developers are thus free to encode the knowledge in a factored form, with the expectation that the factored rules will be combined when they are executed by the agent.

3.4 Improving Situation Interpretations

Accurate interpretations of the rapidly evolving battlefield situation is key to a Soar/IFOR agent's successful task performance. One important component of such an interpretation is accurate tracking of an opponent's ongoing actions, to infer her higher level goals, plans or behaviors. For instance, a Soar/IFOR agent cannot actually observe a missile, but needs to infer a missile firing based on the opponent's maneuvers, as shown in Figure 1. Here, the Soar/IFOR agent is piloting the dark-shaded aircraft and its opponent the light-shaded one. In Figure 1-a the two aircraft are on collision course—if they fly straight they will collide at the point shown by x. After reaching her missile firing range, the opponent turns her aircraft to point at the Soar/IFOR agent's aircraft (see Figure 1-b). In this situation, the opponent fires a missile. She then turns 45-degrees—an *Fpole* turn—to provide radar guidance to the missile, while slowing the closure between the two aircraft. The Soar/IFOR agent cannot observe this missile, but based on the opponent's turn to point at its aircraft and the subsequent Fpole turn, it needs to infer that the opponent has fired a missile.

Unfortunately for the Soar/IFOR agents, the hu-

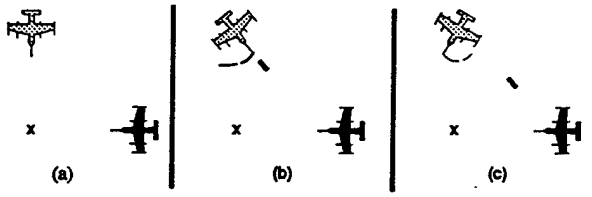


Figure 1: Tracking an opponent's normal missile firing maneuvers. An arc on an aircraft's nose indicates its turn direction. The missile is indicated by –.

man pilots in the STOW-E exercise (see Section 1) were briefed as to what cues Soar/IFOR looks for when interpreting opponent actions, and how they might be able to fool Soar/IFOR by avoiding these cues. They deliberately modified their missile firing behavior to fire missiles while maintaining a 25-degree angle-off (i.e., pointing 25-degrees away from Soar/IFOR agents' aircraft). The Soar/IFOR agents failed to track the missile firing and got shot down. Of course, human pilots are bound to come up with novel variations on known maneuvers, and the Soar/IFOR agents cannot be expected to anticipate them. Yet, at the same time, agents cannot remain in a state of permanent vulnerability—for instance, getting shot down each time the variation of 25-degrees gets used—otherwise they would be unable to provide a challenging and appropriate training environment for human pilots.

The Soar/IFOR agents must adapt their opponent tracking to counter such adaptive behavior on the part of humans. To this end, we are developing the capability to analyze the past combat episodes off-line, and learn from obvious errors. In the above case, the Soar/IFOR agent records in its episodic memory that it got shot down. Its episodic memory of the combat also reveals that it never detected the opponent's missile firing behavior. Simultaneously, however, the episodic memory will note that the agent did face a mysterious maneuver that it was unable to track (corresponding to the missile firing with a 25-degree angle-off). Based on this episodic memory, the agent can learn that the human pilot can fire a missile from a 25-degree angle-off.

4 Practical Aspects of Using Chunking

Given the Soar/IFOR agents' real-world environment, the costs and benefits of chunking have to be evaluated from a practical perspective. The key question here is: Do the benefits of chunking outweigh its costs as it stands *today*? In this regard, the following factors need to be taken into account:

1. The Soar/IFOR agents' current knowledge is already encoded in a highly optimized form, so that they can rapidly respond to opponents' maneuvers. It is difficult for chunking to improve upon such decisions, other than to reorganize the encoded knowledge somewhat, as described above.
2. The agents' current knowledge is the result of extensive knowledge acquisition sessions. Some of the tactical knowledge gained from these sessions is highly sophisticated and a result of careful analysis of the capabilities of the opposing forces. It is difficult for chunking techniques to reconstruct, much less improve on, this expertise.
3. Chunks learned are sometimes highly specific—their conditions refer to the agent's current situation in terms of the value of its altitude, speed, range from an opponent, etc. Such chunks do not transfer (apply) to other similar situations, thus reducing the effectiveness of chunking.
4. The learning process itself can incur development overhead. Modifications to agent code can invalidate previously created chunks. Thus as the agents are modified, training sessions must be run repeatedly in order to produce an up-to-date set of chunks.

The above practical issues in applying chunking, combined with our earlier observations regarding the lack of overall speedups, implies that on-line chunking has to be very carefully applied, if at all, in service of speedups. We find it expedient to turn chunking on when the agents are making certain types of decisions, and turn it off elsewhere.

5 Long-Term Prospects

As development of Soar/IFOR proceeds, new opportunities continue to present themselves for making more extensive use of machine learning, and to employ existing learning abilities in new ways. Episodic memory is a good example of the latter: once an agent has the ability to remember previous episodes, a variety of possibilities for learning from those episodes present themselves. As the added capabilities afforded by machine learning accumulate, and the costs associated with learning are mitigated, the benefits stemming from learning are expected to dominate the costs to a greater and greater extent.

There is reason to believe, in fact, that eventually further improvement in performance of Soar/IFOR agents will only be achievable by means of machine learning. As long as the decision making of Soar/IFOR agents is governed by fixed rules, wily human opponents will learn ways of gaining advantages over the agents. This will be especially true if and when these agents are integrated into training devices that are used on a routine basis. If current work on enabling Soar/IFOR to learn from experience can be applied to a range of situations and scenarios, then human trainees will find simulations to be continually challenging, and able to put their tactical skills fully to the test.

Acknowledgement

We gratefully acknowledge Paul Rosenbloom's comments on this paper, as well as the contribution of the other members of the team involved in the creation of the Soar/IFOR agents, including John Laird, Randolph Jones, Karl Schwamb, and Frank Koss. This research was supported under contract N00014-92-K-2015 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Research Laboratory (NRL) to the University of Michigan, via a subcontract to USC; and under contract N66001-95-C-6013 from ARPA and the Navael Command and Ocean Surveillance Center, RDT&E division (NRAD).

References

[1] R.B. Calder, J.E. Smith, A.J. Courtemanche, J.M.F. Mar, and A.Z. Ceranowicz. ModSAF behavior simulation and control. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pages 347–359, Orlando, FL, March 1993. Institute for Simulation and Training, University of Central Florida.

[2] R.B. Doorenbos. Matching 100,000 learned rules. In *Proceedings of the National Conference on Artificial Intelligence*, pages 290–296, Menlo Park, CA, August 1993. AAAI.

[3] W.L. Johnson. Agents that learn to explain themselves. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1257–1263, Seattle, WA, August 1994. AAAI, AAAI Press.

[4] J.E. Laird, A. Newell, and P.S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[5] S. Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2–3):363–391, 1990.

[6] T. M. Mitchell, Keller R. M., and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[7] P. S. Rosenbloom and J. E. Laird. Mapping explanation-based generalization onto soar. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 561–567, 1986.

[8] K.B. Schwamb, V.F. Koss, and D. Keirsey. Working with ModSAF: Interfaces for programs and users. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavior Representation*, pages 395–399, Orlando, FL, May 1994.

[9] V.J. Shute and J.W. Regian. Principles for evaluating intelligent tutoring systems. *Journal of Artificial Intelligence in Education*, 4(2/3):245–273, 1993.

[10] M. Tambe, W.L. Johnson, R.M. Jones, F. Koss, J.E. Laird, P.S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. To appear in *AI Magazine*, Spring 1995.

[11] M. Tambe, A. Newell, and P. S. Rosenbloom. The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5(3):299–348, 1990.

Biographies

W. Lewis Johnson is a project leader at USC/ISI and a research assistant professor in the USC Department of Computer Science. Dr. Johnson received his A.B. degree in Linguistics in 1978 from Princeton University, and his M.Phil. and Ph.D. degrees in Computer Science from Yale University in 1980 and 1985, respectively.

Milind Tambe is a computer scientist at USC/ISI and a research assistant professor with the computer science department at USC. He completed his undergraduate education in computer science from the Birla Institute of Technology and Science, Pilani, India in 1986. He received his Ph.D. in 1991 from the School

of Computer Science at Carnegie Mellon University, where he continued as a research associate until 1993.

Planning in the Tactical Air Domain*

Randolph M. Jones, Robert Wray, Michael van Lent, and John E. Laird
Artificial Intelligence Laboratory
University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109-2110
{rjones,wrayre,vanlent,laird}@eecs.umich.edu

Abstract

TacAir-Soar is a reactive system that uses recognition-driven problem solving to plan and generate behavior in the domain of tactical air combat simulation. Our current research efforts focus on integrating more deliberative planning and learning mechanisms into the system. This paper discusses characteristics of the domain that influence potential planning solutions, together with our approach for integrating reactive and deliberative planning.

TacAir-Soar (Jones *et al.* 1993; Rosenbloom *et al.* 1994) implements artificial, intelligent agents for use in tactical flight training simulators. The overall goal of the project is to create automatic agents that generate behavior as similar as possible to humans flying flight simulators. These agents will help provide relatively cheap and effective training for Navy pilots.

In order to accomplish this task, we need not only to acquire and encode a large amount of complex knowledge, but also to address a number of core research issues within artificial intelligence. Not the least of these issues is the ability for the agent to plan its activities appropriately, and to acquire efficient and effective new behaviors as a consequence of planning.

We are investigating the hypothesis that a variety of appropriate behaviors can arise from a system with a small, organized set of cognitive mechanisms as it interacts with a complex environment. Thus, the primary thrust of our research relies on *integration* in a number of different forms. Reactive behavior generation must be integrated with goal-directed reasoning and planning. These in turn must be integrated with other cognitive capabilities, such as situation interpretation, natural language understanding and generation, plan recognition, planning, etc. Rather than combining distinct modules for execution, planning, and learning, we are attempting to integrate all of these capabilities within a single control scheme. Thus, planning becomes simply another form of execution, which must interact with other knowledge in order to generate appropriate behavior. Learning occurs as a side effect of execution, manifesting itself in different ways depending on the particular tasks being executed. Because of the incremental, dynamic, and complex nature of behavior generation in the tactical air domain, learning must also be incremental, fast, and able to capture the complexities of goals and actions.

The current version of TacAir-Soar combines reactive and goal-driven reasoning to create what we call *recognition-driven problem solving* (Tambe *et al.* 1994). The system contains a large set of rules that fire as soon as their conditions are met, without search or conflict resolution. Some of these rules respond to immediate changes in sensory inputs, while others respond to higher-level interpretations of those changes and goals that the system posts for itself. As an example, TacAir-Soar may observe a series of readings about a contact on its radar, and conclude that the contact is an aggressive enemy aircraft. Thus, the system posts a goal of intercepting the aircraft, which involves maintaining a collision course. The actual heading of TacAir-Soar's aircraft will change every time the collision course changes. This paradigm for behavior generation is similar to *reactive planning* in the spirit of Firby's (1987) RAP planners. That is, the system does not perform any search to determine the best course of action, and it does not plan in terms of predicting future states of the environment.[1] It also

Thanks to Paul Rosenbloom, Lewis Johnson, Soowon Lee, Frank Koss, and the reviewers for their comments on earlier drafts of this paper. In addition, this research has benefited enormously from the combined efforts of the Soar-IFOR group. The research is supported by contract N00014-02-K-2015 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Research Laboratory.

[1] TacAir-Soar agents do some prediction, but it is part

computes its behavior dynamically, rather than generating a declarative plan that is later interpreted. Part of our current research effort is to equip TacAir-Soar with a deliberative planning component that separates planning from normal execution by projecting future possible states and searching through them to decide on appropriate courses of action.

Of the following three sections, the first provides a short motivation for the usefulness of deliberative planning in the tactical air domain. The second lists a number of characteristics of the domain that have a significant impact on how planning must occur. These characteristics have been discussed in various earlier work on planning, but our work will address all of them together and attempt to provide a planning solution that naturally integrates into recognition-driven problem solving. The final section sketches potential solutions for deliberative planning. These solutions are suggested by a combination of the characteristics of the domain, our desire for a fully integrated system, and the problem-solving and learning paradigms provided by the Soar architecture.

Advantages of Deliberative Planning

As mentioned previously, the overall goal for the TacAir-Soar system is to generate human-like behavior within the simulated environment. One hallmark of human behavior is flexibility in the face of new situations. The current system has been equipped with a large knowledge base of tactics, vehicle dynamics, weapons characteristics, etc., and this allows the system to generate a wide variety of behaviors in response to different situations, missions, and goals. One approach to this type of domain has been to attempt to capture every possible situation that an agent may encounter in a recognition rule (e.g., Bimson *et al.* 1994). However, even if such an approach is possible, it would require extensive work on the knowledge base every time the domain changes a bit (for example, if new aircraft or missiles are developed).

In response to this difficulty, an agent must detect when it does not have suitable knowledge to react to a particular situation, and use its planning capabilities to generate appropriate actions based on more fundamental knowledge. This requires the agent to integrate deliberative planning with its current recognition-driven reasoning mechanisms. Naturally, we also expect the agent to learn from its planning episodes, generating new rules for future similar situations.

TacAir-Soar will do much of its planning "in the air," where there are tight restrictions on time, thus limiting the learning opportunities. However, human pilots often learn by flying real or simulated scenarios, and then debriefing the scenarios on the ground. By going back over each step of the scenario, the pilot can identify successes and failures, consider alternative courses of action, and take more time to evaluate various possible outcomes. Automated agents have also been demonstrated to benefit from such self-explanations (VanLehn, Jones, & Chi 1992). In addition, Johnson (1994a; 1994b) has presented a debriefing facility, in which TacAir-Soar agents can explain their actions after a scenario, and consider some hypothetical alternatives. The deliberative planning mechanism should expand on this approach and allow the system to learn from the debriefing experience. In addition, we intend the same planning mechanism to be used for planning both in the dynamic environment of an engagement and the calm, slow-paced environment of a debriefing session. Naturally, when the agent has more time to plan, the quality and quantity of effective learning should increase, but this will be due to the dynamics of the planning situation, not because of any differences in the planning and learning mechanisms.

Planning Issues for Tactical Flight

This section focuses on the specific aspects of the tactical air domain that have a significant impact on how planning should be carried out. There are five particular characteristics that set the domain apart from traditional domains used in planning research.

Interaction of Domain Goals

The current version of TacAir-Soar knows about almost 100 different types of goals, and many of these interact with each other. For example, there are times when an agent wants simultaneously to fly toward a target, evade an incoming missile, and maintain radar contact with another aircraft. This presents the traditional problem of planning for goal conjuncts (Chapman 1987; Covrigaru 1992). However, we must trade off the intensive search that can be involved in this type of planning with the dynamic and uncertain nature of the task (discussed below). Other researchers (e.g., Cohen *et al.* 1989; Veloso 1989) have suggested methods for planning about conjunctive goals in real time, and we hope to borrow from these approaches in our own efforts.

Two primary elements of conjunctive goal planning are detecting a goal interaction and then finding a way to deal with the interaction. Within TacAir-Soar, interactions will generally be detected when conflicting output commands are sent to the simulator (e.g., to come to two different headings) or when goal con-

of normal behavior generation, and not something that is learned about for decision making.

straints are incompatible (e.g., turning away from a target while also maintaining a radar lock). In general, there will be two methods for dealing with such goal interactions. Some goals can be achieved conjunctively (perhaps not as efficiently as if the goals were independent), but sometimes it will be necessary to suspend certain goals temporarily when goals of higher priority (such as evading an incoming threat) conflict with them.

Dynamic, Real-Time Environment

As suggested above, TacAir-Soar cannot generally assume that it has ample time to plan. An agent may be planning an intercept course to a target when it detects an incoming missile. In this case, the agent must interrupt its planning in order to react in a timely fashion. As a slightly different case, the situation may change so rapidly that the conditions that initiated planning may become obsolete before planning is completed. For example, the agent may begin planning which type of weapon it should employ against a target, only to find it destroyed by some other participant in the engagement. In both of these situations, the system should cease its planning activity, even if it did not find a result. Reactive planning systems (e.g., Agre & Chapman 1987; Firby 1987; Kaelbling 1986), and TacAir-Soar's recognition-driven problem solving address some of these issues by dynamically changing goals and behaviors as the environment changes. The next challenge is to integrate deliberative planning with dynamic reasoning in a smooth way.

Large State Representation

A further characteristic of the domain is that it involves rather large representations of the agent's current situation. The state representation includes information about various vehicle and weapon types, sensor information (from visual, radar, and radio sources), the agent's current mission goals, other "mental" annotations, and interpretations of the state, actions, and goals of other agents. For normal recognition-driven problem solving, the situated TacAir-Soar agent simply reacts to various features in this large state by generating actions or posting new goals or new interpretations of the situation.

The size of the state can impact deliberative planning in three ways. First, any time the agent wishes to plan, it must construct a copy of its current state representation. It can then manipulate this copy without changing its actual representation of the world or issuing real behaviors. Second, separating the two state representations allows the system to generate low-level reactions in response to one state while planning with the other. Because it takes some time to create this mental planning state, the agent should copy only the necessary information for planning and no more. Finally, some of the state information will be important to the current plan, while other information will be less important or totally irrelevant. It is not desirable for the agent to reason about portions of the state that have no bearing on the current decision. Thus, decisions about how much state to copy will have an impact on learning and the generality of new behaviors.

Planning in the Face of Uncertainty

A key feature of the tactical air domain is that there is generally a large number of participants in any given scenario. Some research (e.g., Georgeff 1984) has focused on this problem, and it naturally will have a strong effect on how TacAir-Soar can interpret and predict the consequences of its actions while planning. Anticipating the actions of cooperating agents may not be too difficult, because there exist social engagements and standard operating procedures between agents that cooperate. Predicting the future actions of competing agents is somewhat more difficult, and relies in part on recognizing the plans and goals of those agents (Tambe & Rosenbloom 1994; Wilensky 1981).

Given the unpredictable nature of modeling other agents, it is most appropriate for TacAir-Soar to create completable plans (Gervasio & DeJong 1994), in order to react appropriately to future actions by other agents. Contingency plans (Warren 1976) might also be useful, but these are generally expensive to generate. In a sense, TacAir-Soar's current knowledge base consists of a large completable plan, and such planning is consistent with our desire to integrate the current recognition-driven problem-solving structure with deliberative planning. The results of deliberative planning should be completable, reactive plans that the agent can execute and adapt in response to the dynamics of the environment.

Termination of Planning

As we have already mentioned, available time will have a large impact on how long any planning activity can continue. However, termination of planning is also influenced by when results can be produced. Most traditional planners have small sets of explicit, well-defined goals, and a precise evaluation function, so they can plan until a method is found to achieve their goals. Within the tactical air domain, there are many different types of goals, and different degrees to which they can be achieved. As an example, if an aircraft has the mission to protect its aircraft carrier, it may produce the goal of destroying an incoming attack aircraft. After the engagement has proceeded, the agent may find

itself drifting from the carrier it is supposed to protect. At this point, it may decide that it has completed its mission by "scaring off" the threat, without actually destroying it, and it would be more dangerous to continue than to return to its patrol position.

The combination of limited reasoning time and ill-defined goals provides a further complexity for planning. The question is how far the planning process should continue, and when evaluation should take place.

Solutions for Deliberative Planning

These characteristics all have an impact on how planning can occur in an intelligent agent. Many of these issues have been addressed to some extent in previous research, but we hope to build an integrated system that addresses all of them. This section describes our preliminary efforts to develop an integrated planning solution that addresses all of the complexities of the domain. It begins with a discussion of the overall integrated framework, and then describes specific ideas for each of the planning issues.

Integrated Planning, Learning, and Execution

Our commitment to an integrated system began with our selection of the Soar architecture (Laird, Newell, & Rosenbloom 1987) as the platform for development. Soar provides an ideal basis for recognition-driven problem solving, and naturally supports the integration of execution, planning, and learning (Laird & Rosenbloom 1990).

Readers familiar with Soar will recall that all reasoning and behavior generation takes place in problem spaces, through the deliberate selection of operators. A fair amount of research on traditional planning within Soar (e.g., Lee 1994; Rosenbloom, Lee, & Unruh 1992) also organizes planning knowledge as sets of problem spaces. Problem spaces are collections of knowledge that address subgoals, which arise in response to a lack of knowledge in a particular situation. A typical example for planning occurs when an agent has a number of candidate actions to take, but does not have the knowledge to decide between them. For example, a pilot must decide which type of weapon to employ against a target, given the current mission and circumstances of the environment. After planning knowledge (e.g., a mental simulation of the alternatives) suggests an ordering, the automatic learning mechanism summarizes search in the problem space into individual rules ("chunks") that will apply in future similar situations.

We should stress the point that the natural representation for a plan within TacAir-Soar is not a declarative script of actions. Rather, a plan is a collection of recognition-driven rules and operators that apply opportunistically in response to particular patterns of sensor values, interpretations, and goals. Thus, in a sense, TacAir-Soar will never be learning entire plans, but it will be repairing or completing the general plan composed of all of its recognition rules.

Addressing Domain Issues

This integrated framework suggests possible solutions for planning that also address the issues presented earlier. To begin with, the high degree of interaction between goals suggests criteria for both triggering and evaluating new plans. Previously, we suggested that planning occurs when TacAir-Soar does not have the reactive knowledge necessary to choose between competing actions. This can be generalized to initiating planning any time the system detects an interaction between goals that it does not know how to handle. Covrigaru (1992) and Lee (1994) have investigated planning methods within Soar to address interactions between different types of goals. Evaluation of potential plans will be based on the resolution of individual interactions—as opposed to, for example, planning exhaustively until all interactions are resolved. As the agent develops responses to individual interactions, it can learn partial planning results in the form of new recognition rules.

These partial results also address the dynamic characteristics of the domain. Such planning will integrate smoothly with normal behavior generation because every planning episode will cause the system to learn *something*. If it is not something that completely resolves the current situation, it should at least allow the planning process to resume later without having to start over. Thus, particular planning efforts can be temporarily suspended (or perhaps abandoned entirely) without having been a total waste of time. When the system has ample time to plan (such as in a debriefing session), it is not clear whether the planning process will need to be qualitatively different. Presumably, the system will still be able to use its incremental planning techniques, but generate better quality plans because it has more time to evaluate and resolve interactions.

Also in response to to the dynamic domain, our initial efforts with TacAir-Soar have addressed the issue of integrating planning with execution. Many of the system's actions can apply without regard for whether the system is currently planning. For any aspects of the current situation that do not depend on the current planning activity, the system continues to generate behavior independent of other processing.

Because of TacAir-Soar's large state representation, we have adopted high-level, qualitative descriptions that summarize direct sensor readings, thereby reducing the amount of information that must be copied. In addition, the system attempts to make intelligent decisions about the portions of the state it cares about. These decisions are based on a static analysis of the domain knowledge, as well as dynamic reasoning based on the current situation. This allows the system to limit the amount of work it does in creating a mental copy of the state, which has been our primary concern in preliminary work on planning.

Our hope is that this approach will also aid the system in reasoning in an uncertain environment. As we have discussed, an appropriate response to this issue is to generate completable plans. In TacAir-Soar's terms, we wish to learn new rules for posting general goals, allowing the specific situation at execution time to dictate the precise actions that should be taken to satisfy those goals. Thus, a further aim for setting up a mental state for planning is to abstract away details that can be filled in by the situation later.

Finally, the criteria for terminating the planning process arise in part from the solutions we have already discussed. If there is time to plan exhaustively, the system will generate solutions for all the goal interactions it detects. Because the system returns incremental results as it plans, it is not as important for it to determine a fixed stopping criterion. If planning must be suspended temporarily, the partial planning results should allow planning to resume from where it left off. Finally, as we have mentioned, the system is able to generate behavior simultaneously with planning in many situations, so planning will not have to be interrupted until it is actually finished.

Summary

Simulated tactical air combat is an ideal, real domain for developing and testing new planning methods. The complexities of the task require us to focus on a number of planning issues that can be safely ignored in traditional planning domains. Although many of these issues have been addressed to some extent in the planning literature, we plan to provide an integrated solution to all of them. We have begun creating a system that smoothly integrates reactive and deliberative planning within the recognition-driven problem solving framework. Although our efforts with the deliberative planning component are young, our initial experiences have been encouraging. Hopefully, the complexities and real-time demands of the tactical air domain will lead us to a system that can model a continuum of planning processes from purely reactive to knowledge intensive and deliberate.

References

Agre, P. E., and Chapman, D. 1987. Pengi: An implementation of a theory of action. In *Proceedings of AAAI-87*, 123–154. Menlo Park, CA: AAAI Press.

Bimson, K.; Marsden, C.; McKenzie, F.; and Paz, N. 1994. Knowledge-based tactical decision making in the CCTT SAF prototype. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, 293–303.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32:333–377.

Cohen, P. R.; Greenberg, M. L.; Hart, D. M.; and Howe, A. E. 1989. Understanding the design requirements for agents in complex environments. *AI Magazine* 10(3):32–48.

Covrigaru, A. 1992. *Emergence of meta-level control in multi-tasking autonomous agents.* Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, University of Michigan.

Firby, R. J. 1987. An investigation into reactive planning in complex domains. In *Proceedings of AAAI-87*, 202–206. Menlo Park, CA: AAAI Press.

Georgeff, M. 1984. A theory of action for multiagent planning. In *Proceedings of AAAI-84*, 121–125. Menlo Park, CA: AAAI Press.

Gervasio, M. T., and DeJong, G. F. 1994. An incremental learning approach for completable planning. In *Machine Learning: Proceedings of the Eleventh National Conference*, 78–86. San Francisco, CA: Morgan Kaufmann.

Johnson, W. L. 1994a. Agents that explain their own actions. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, 87–95.

Johnson, W. L. 1994b. Agents that learn to explain themselves. In *Proceedings of AAAI-94*, 1257–1263. Menlo Park, CA: AAAI Press.

Jones, R. M.; Tambe, M.; Laird, J. E.; and Rosenbloom, P. S. 1993. Intelligent automated agents for flight training simulators. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, 33–42.

Kaelbling, L. P. 1986. An architecture for intelligent reactive systems. In *Proceedings of the Workshop on Planning and Reasoning about Action*, 235–250.

Laird, J. E., and Rosenbloom, P. S. 1990. Integrating execution, planning, and learning in Soar for external

environments. In *Proceedings of AAAI-90*, 1022–1029. Menlo Park, CA: AAAI Press.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33:1–64.

Lee, S. 1994. *Multi-Method Planning.* Ph.D. Dissertation, Department of Computer Science, University of Southern California.

Rosenbloom, P. S.; Johnson, W. L.; Jones, R. M.; Koss, F.; Laird, J. E.; Lehman, J. F.; Rubinoff, R.; Schwamb, K. B.; and Tambe, M. 1994. Intelligent automated agents for tactical air simulation: A progress report. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, 69–78.

Rosenbloom, P. S.; Lee, S.; and Unruh, A. 1992. Bias in planning and explanation-based learning. In Chipman, S., and Meyrowitz, A., eds., *Machine Learning: Induction, analogy and discovery.* Norwell, MA: Kluwer.

Tambe, M., and Rosenbloom, P. S. 1994. Event tracking in complex multi-agent environments. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, 473–484.

Tambe, M.; Jones, R. M.; Laird, J. E.; Rosenbloom, P. S.; Schwamb, K.; and Koss, F. 1994. Intelligent agents for interactive simulation environments. Manuscript in preparation.

VanLehn, K.; Jones, R. M.; and Chi, M. T. H. 1992. A model of the self-explanation effect. *Journal of the Learning Sciences* 2:1–59.

Veloso, M. M. 1989. Nonlinear problem solving using intelligent casual commitment. Technical Report CMU-CS-89-210, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Warren, D. H. D. 1976. Generating conditional plans and programs. In *Proceedings of the AISB Conference*, 344–354.

Wilensky, R. 1981. *Inside computer understanding: Five programs plus miniatures.* Hillsdale, NJ: Lawrence Erlbaum.

Multiagent Coordination in Distributed Interactive Battlefield Simulations*

John E. Laird, Randolph M. Jones, and Paul E. Nielsen
Artificial Intelligence Laboratory
University of Michigan
1101 Beal Ave.
Ann Arbor, MI 48109-2110
laird@umich.edu

Introduction

On November 4–7, 1994, the Department of Defense held an operational exercise called STOW-E, involving over 1,800 entities in a virtual battlefield, making this one of the largest applications of real time, multi-agent simulation. The participants included both humans (in simulators and specially instrumented vehicles) and computer generated forces, interacting in real-time, unscripted, realistic engagements. By 1997, DOD plans to hold a virtual theater-level war involving up to 50,000 entities. These simulations provide a cost-effective and flexible environment for training, mission rehearsal, and tactics development. The computer forces are implemented via a spectrum of approaches, from aggregate forces generated by wargames, to human managed semi-automated forces (SAFORs), to completely autonomous intelligent forces (IFORs). The computer forces dominate in terms of sheer numbers, with at least 10 times as many computer generated forces as human forces.

Our interest is in the development of IFORs, computer agents with the ability to participate fully in all aspects of the simulated battlefield. The Soar/IFOR consortium, involving the University of Michigan, Information Sciences Institute of the University of Southern California, and Carnegie Mellon University, is developing IFORs for all military air missions: air to air combat, air to ground attacks, air supply, anti-armor attack, etc. IFORs must have many capabilities to be successful: real-time reactivity, goal-directed problem solving, planning, large bodies of knowledge, and they must coordinate their behavior with other friendly forces. Furthermore, to be useful and effective in training and tactics development, the tactical behavior of our agents must be humanlike. We have demonstrated the feasibility of developing IFOR agents (Rosenbloom *et al.* 1994), and our agents fully participated in STOW-E, flying air-to-air, ground attack, and surface attack missions against human and computer generated forces.

Within this domain, coordination is one of the most important determiners of success. A single unit has only limited ability to sense its environment directly, and has only limited ways in which it can act. Through coordination of sensing, multiple agents can share their knowledge about the environment, thus making their actions far more effective. Through coordination of their actions, multiple agents can avoid conflicting action and they can perform actions that no single agent can perform alone, such as mutual defense. The problem is how to get many different agents, in different physical locations, with different models of the environment, with different physical abilities, and possibly different short-term goals, to work together to achieve their common long term goals.

Previous work in computer-generated forces (Calder *et al.* 1993) has not extensively modeled the coordination of individual forces. In the majority of cases, either an omniscient human or computer agent provides the appearance of coordination through low-level monitoring and controlling of individual agents. When tight coordination of behavior of a small unit is required, such as a section of planes flying in formation, the aggregation is treated as a single unit. Instead of attempting to represent the communication and coordination of the individual planes, behavior is generated for the section as a whole and then specialized for the individual unit (Rao *et al.* 1994). Thus, individual units are not faced with integrating coordination activities with their own goals, nor do they need to communicate with other units.

Our approach is straightforward. We model the command and control methods currently in use by military organizations. Thus, our agents directly model the performance of humans: there is a one to one mapping between our agents and humans. Our agents have the same limits in perception and action that a human would have, and they must coordinate their behavior just as humans do, through shared knowledge and communication. Some of the advantages of this approach include:

1. Coordinated behavior is more realistic. Our agents coordinate based on shared doctrine, shared mis-

*This research was supported under contract N00014-92-K-2015 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Research Laboratory.

sions, and explicit communication. Explicit communication requires time to transmit and interpret, and is open to mis-interpretation, jamming, etc. By independently modeling each entity (instead of a group as a whole), our agents can take the initiative when appropriate.
2. Coordinated behavior should be easier for humans to understand because there is explicit communication to monitor.
3. Coordination is possible between human and computer forces because the communication is modeled on human communication.

In building our agents, we discovered that some of the issues that have plagued previous research in multi-agent coordination do not arise in this domain. First, coordination is possible *without* the addition of special purpose "architectural" capabilities (such as the generation and transmission of partial-global plans (Durfee & Lesser 1987; Durfee 1988)). An architecture designed to support general intelligent agents — such as Soar (Laird, Newell, & Rosenbloom 1987) — appears to be sufficient. Coordination does require large bodies of knowledge and inference (it is a knowledge-level capability (Huffman, Miller, & Laird 1993)), but these need not be specialized except in content. Second, our agents do *not* need to carry out protracted negotiations (Rosenschein 1993; Smith 1980; Sycara 1989), "reason about the *processes of coordination among the agents*" (Bond & Gasser 1988), or dynamically construct complex models of those agents (Ephrati & Rosenschein 1992). Because our agents are designed to work within the military's well establish hierarchy of command, control, and communication, and because our agents are "experts" for their tasks, negotiation, runtime reasoning, and complex internal models of friendly agents can be "compiled" out, with just the knowledge of how and when to coordinate remaining. Our agents require only very limited information about other agents, such as locations, call signs, radio frequencies, and positions within the command hierarchy.

The goal of this paper is to demonstrate the sufficiency of our simplier approach for a real-world application, and to analyze the complexity of coordination required in this domain.[1] We begin by presenting a scenario that illustrates the coordination required in this domain. We then analyze the required coordination along three dimensions: the organizations of the agents, the type of activities that are coordinated, and finally the sources of knowledge that support coordination. Next we identify the general capabilities that are required to support coordination and how they are realized in our underlying architecture of choice (Soar). We conclude with a discussion of the limits our approach.

[1]This is an extension of our earlier work on air-to-air coordination (Laird, Jones, & Nielsen 1994).

Example Scenario

Our agents include pilots of fighter and air-to-ground attack planes, and a variety of controllers that provide mission and routing information to the planes. We attempt to realistically model current military doctrine and tactics. Our sources include unclassified military documents, books, extensive interviews with former U.S. Navy pilots, and observations of U.S. Navy pilots training in real aircraft and in military flight simulators. Our agents are built within TacAir-Soar (Rosenbloom *et al.* 1994), our generic name for agents that fly simulated fixed-wing aircraft developed within Soar. Our agents' simulation environment is based on the DIS protocol (steering committee 1994). The agents interact with the DIS world through ModSAF (Calder *et al.* 1993), which provides simulations of vehicle dynamics, sensors, and weapons. DIS (and ModSAF) support distributed, interactive, real-time simulation for ground, surface, and air entities. For example, in STOW-E, our planes engaged both humans in simulators and SAFOR computer generated forces. Our planes fired simulated exocet missiles at a real ship (the Hue City) that was participating in the simulation through special instrumentation, bombed a virtual bridge, shot down humans in simulators, and were shot down by humans in simulators and once by a virtual surface-to-air missile. Each of our agents is an independent Soar system situated in its own virtual vehicle (such as an F-18), and is restricted to perceiving what would be available to a human in such a vehicle (via radar and vision). Communication between agents takes place via simulated radios using messages that approximate the messages sent by humans.

Consider the scenario in Figure 1 in which two fighter planes (F-18's) are flying as a section on an air-to-ground mission. This is similar to a mission flown by Soar agents during STOW-E, but has been expanded for expositional purposes to include a broader variety of coordination types (all of which are implemented in our agents). The original goal of the mission is to bomb Target 1. Once the planes are airborne, they join up into a prebriefed formation (at time 1), and start on their prebriefed flight plan (Elmer to Cougar to Wanda to Target 1). The lead (F1) controls the section and makes all section-level mission decisions, as well as flying his own plane. The wingman (F2) flies so as to maintain the current formation.

While flying their route, the lead checks in with a controller (the TACC) to receive permission to enter the combat area, and to receive possible changes in routing information. The controller may request authentication to verify that the plane is friendly. The job of the controller is to verify that planes are where they belong, to perform air traffic control to avoid collisions (usually by assigning different routes and altitudes), and to relay commands from other command entities.

Let us assume that an E-2C (at time 2) informs the

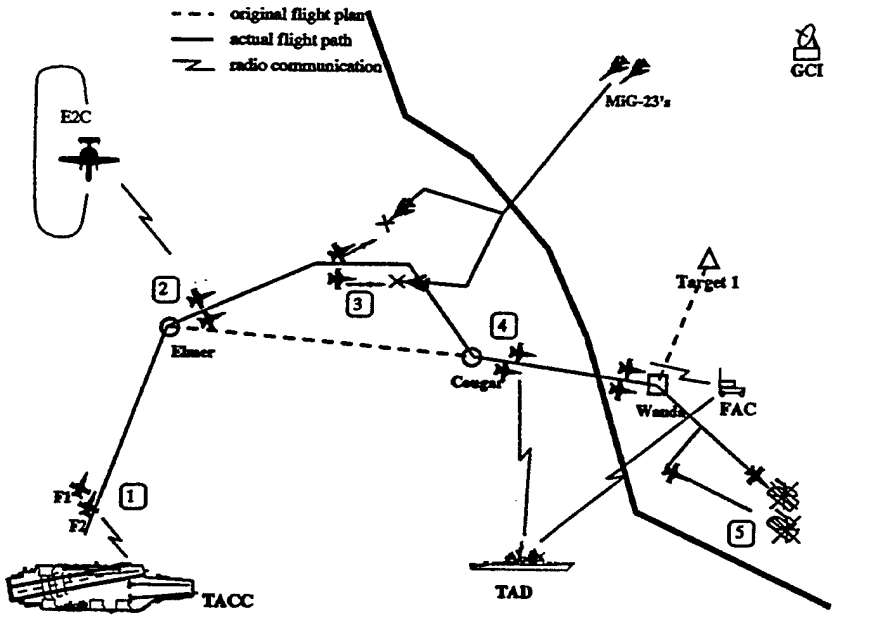


Figure 1: Example Scenario

planes of an approaching threat (the MiG-23's). Usually there would be other planes available to deal with the MiG's, but this allows us to illustrate some important types of coordination. The MiG-23's must depend on a ground controller (GCI), which has a much more powerful radar to guide them toward the engagement by giving them bearing and range information of the F-18's. Similarly, the F-18's will get information about the MiG's from the E-2C. However, once the F-18's have the MiG's on their own radar, and they have been cleared to engage by their controller, they will prosecute the engagement on their own, receiving information from the controller only when they request it (such as if they lose radar contact).

During the engagement, the lead of the F-18's communicates to the wingman to change formation to one that provides more mutual support, and to "sort" the MiG's so that they each have a separate target. Similarly the MiG's communicate to perform their tactical maneuver (called a pincer). In this engagement, we assume that both MiG's are destroyed (at time 3). In general, the F-18's would jettison their bombs prior to the engagement to increase their maneuvering ability, but we will assume that they did not and continue their ground attack mission.

Once the F-18's destroy the MiG's, they head back to their next waypoint (Cougar). At about this time, the forward air controller (FAC) comes under attack by enemy tanks. The FAC calls to another controller (the TAD) and requests an air strike. The TAD contacts the F-18's (at time 4) and gives them a new mission. The lead of the F-18's must then plan their final attack altitude and geometry from the "initial point" (Wanda) to the target and communicate it to the wingman. As they approach the initial point (wanda), the lead communicates the mission to the FAC to verify the target, etc. Once the FAC verifies the mission and visually sites the planes, the FAC sends them a "cleared hot" message to attack the target. The F-18's perform a 90-10 maneuver (planned earlier by the lead) to provide separation during the final bombing run. Since the tanks are moving, the F-18's must visually acquire them, modify their approach, and drop their bombs (at time 5). They then exit the attack area and fly back on their egress route (not shown).

Although, our agents embody all of the reasoning and communication required in this scenario, they do not embody all that humans use in the complete range of air-to-air and air-to-ground missions. For example, our planes only fly in groups of two (sections), not in groups of three or four (divisions). Also, a forward air controller can mark a target for a plane by using a flare, a beacon, or a laser. In addition, our E-2C agents do not direct planes to specific air targets. Currently they only provide contact information that our agents use to make their own decisions. We plan to implement all these types of coordination in the near future.

Coordination Analysis

The purpose of this analysis is to demonstrate the diversity of coordination being performed by our agents across a variety of dimensions.

Coordination Organization

The previous example illustrates the three organizational structures of coordination used by the military and our agents. In all cases, a section or a plane is controlled by another entity (lead, or controller), but the individual agents still have significant autonomy and responsibility for their own actions. The command

structure is relatively static, and a section is in contact with only a single controller at a given time. The current controller for a section is determined by the mission briefing, or by explicit communication from another controller.

- Master slave: The lead dictates the actions of the section, but the wingman still decides how best achieve and stay in formation. The wingman can become lead if he has better situational awareness or weapons capability.
- Centralized: The GCI and E-2C (as well as the TACC and TAD) can provide information and control for many sections of planes.
- Distributed: The TACC, TAD, and FAC form a distributed control network in which requests for missions are propagated through the network and assigned to sections. The controllers coordinate the activities of multiple fighters by routing them, assigning altitudes, communication frequencies, and attack times.

Types of Coordination

Within the different coordination organizations listed above, the agents coordinate a variety of different activities. The left half of Figure 2 lists the types of coordination found between the lead and wingman in terms of coordinated action, sensing, missions, and section organization. This organization is not represented explicitly within the agents. Most of the coordination occurs in terms of action: flying in formation and employing their weapons. They coordinate in sensing, by directing their radars so that they are not completely overlapping. They explicitly communicate radar and visual sightings. They also coordinate their execution of their mission, and the lead will communicate changes to the mission, current progress in the mission, and intent, such as the decision to intercept enemy planes. Finally, they coordinate the organization of the section.

The right half of Figure 2 shows the types of coordination found between a section and a controller. Here there is no coordination of their joint actions (although the mission coordination provides indirect coordination of the section with other planes). The coordinated sensing is in similar spirit to the coordination within a section, although in this case the controllers have much better radar capabilities. The most involved coordination comes under the mission heading, where the controllers can change almost any aspect of the mission for a section, including the altitude for flying routes, the routes, the controllers the section contacts along the route, the radio frequency to use during the contact, the target location, and the time of the attack. Because the timing of an attack is critical (e.g. it may be coordinated with the ending of an artillery barrage), the controllers also provide a time *hack*, to synchronize everyone's watches. The need to attack at a specific time (+/- ten seconds), forces the planes to adjust their speed dynamically or even go into holding patterns.

Basis of Coordination

In this domain, the key to coordination is knowledge. The agents must know the appropriate techniques and methods for performing their specific tasks, such as maneuvering, sensing, and employing their own weapons. They must know their responsibilities for their current mission, the details of that mission, and who the other agents are that they must interact with. They also must known in general terms when and what to communicate to which agents during the mission, and what to do in response to messages from others. We have identified four different sources of coordination knowledge.

Background Knowledge: Common Doctrine and Tactics. Most of the long-term knowledge in our agents consists of knowledge about how to perform their missions. This includes how to maneuver, sense and employ weapons, but it also includes doctrine and tactics which specify methods and procedures for coordinating with other agents. This doctrine includes specific roles for individuals (such as lead, wingman, TAC, TADD, FAC) and the specific duties to be performed. Thus, there is no need for the lead and wingman to negotiate how to maintain the formation. The wingman just does it. This is a *social contract,* where agents implicitly create coordinated behavior by behaving according to certain prespecified rules (Shoham & Tennenholtz 1992).

The reliance on common background knowledge to support coordination in the military is not surprising. The military has sufficient planning and training time to develop and implement common doctrine and tactics. The individual agents need not determine the best coordination strategy on their own, but can rely on compiled versions of the coordination strategy.

Mission Briefing. Before a mission, the participants are briefed on the tactical situation (such as weather and enemy activity), their responsibilities, and the responsibilities of others. The briefing helps establish specific operational parameters required for coordination, such as the partners of a section, their initial formations, the methods for communication (radio frequencies, call signs), the default radar contract, the default method for sorting enemy planes, any specific tactics the section plans to employ, the waypoints of the mission, the controllers who will be contacted during the mission, the authentication procedures, and so on. Based on the mission briefing, the lead will fill in any details, such as an attack plan, based on the mission and the tactical situation. In our agents, the mission briefing knowledge is not "compiled" into the agent, because it changes from mission to mission. It is a set of parameters that are loaded into our agents before they start their missions.

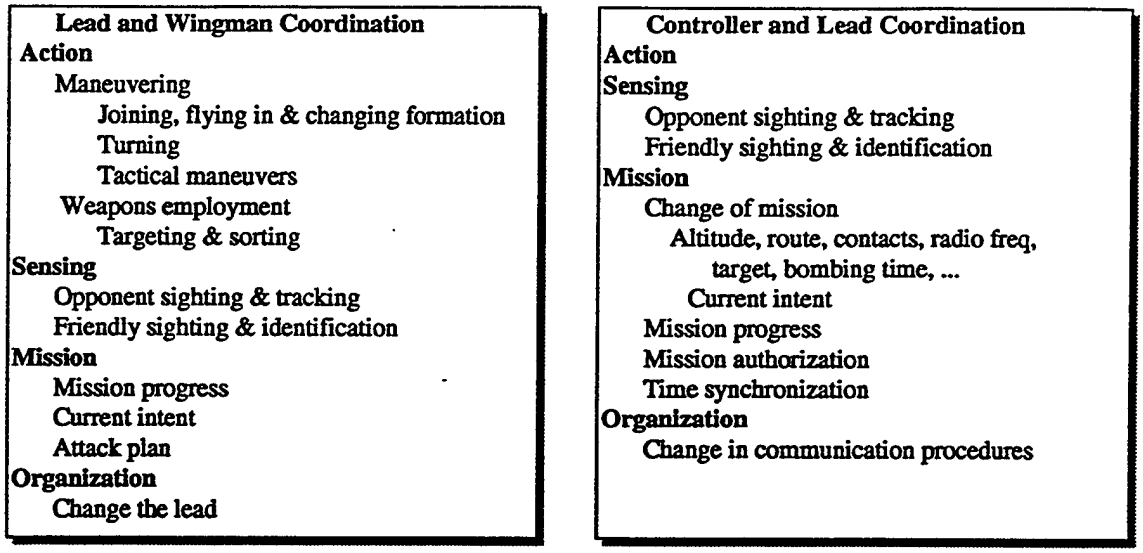


Figure 2: Types of Coordination Between TacAir-Soar Agents.

Observed Behavior. During a mission, the members of a section can directly observe each other's behavior. Thus, behavior alone can be a signal for coordination, as when a lead makes a small turn. In TacAir-Soar, the only use of coordination through observation is when the wing responds to small turns of the lead. This will be expanded when our agents need to fly without radio communication because of jamming or the danger of detection.

Explicit Communication. The most flexible way to coordinate behavior is to explicitly communicate knowledge and goals between two agents. In this domain, it is via radio. We have attempted to replicate the communication used by humans for the missions performed by our agents. There are approximately 70 message templates that our agents can send and receive. These message templates approximate the language used by naval aviators and are easily understandable by humans.

Coordination Capabilities

In this section, we summarize the cognitive capabilities required to support coordination in our agents. This is based on types of coordinated behavior (acting, sensing, mission, etc.), and the methods for sharing knowledge and goals. These capabilities serve as a requirements list for constructing an agent that can coordinate with others in domains such as tactical air combat. For each capability, we describe briefly how it is implemented in TacAir-Soar.

Extensive Knowledge Base. Our approach relies heavily on the fact that the individual agents are experts at performing their missions and interacting with others. Each agent must have an extensive knowledge base that includes all of the tactics and doctrine applicable to its possible roles in the missions in which it will participate. For example, a wingman must have the same knowledge of doctrine and tactics as the lead, so that the wingman can take over when necessary.

TacAir-Soar's knowledge is encoded as rules. Our attack aircraft have over 2600 rules, while our ground-based controllers have over 1800 rules. The doctrine and tactics are encoded within a hierarchy of intertwined goals that are dynamically instantiated based on the current situation and mission.

Parameter-driven Behavior. An agent must not be limited to only one type of behavior, but must be able to perform a variety of activities in coordination with others. In our agents, the mission briefing received before launch and the mission changes received from controllers dynamically determine the goals of our agents. The agent's behavior must be parameterized so that the knowledge relevant to the current mission is used. These may sound trivial, but for some complex missions, the information in the briefing may involve fragments of plans that the agent must integrate into its overall behavior at the appropriate times. Thus, the generators of the agent's behavior must be flexible enough so that they can be modified at any time.

In TacAir-Soar, all mission-related behavior is based on a representation of the current mission that is held in a working memory. This can be examined by the rules that make up its long-term knowledge. The mission is specified at briefing time, but also can be dynamically changed later.

Reactive Execution and Interruptible Processing. A wingman must respond quickly to changes in the lead's behavior. Computer generated forces must

in general be reactive, but coordination also requires that they can interrupt their current goals to process and respond to an urgent message.

In TacAir-Soar, the wingman's main goal is to fly in formation with the lead. Whenever the wingman is out of position, rules fire to propose operators to modify the heading, speed, or altitude. Whenever the wingman receives radio messages from the lead, rules fire to propose operators which in turn perform an action appropriate to the message in the current situation.

Generate and Comprehend Messages. In order to communicate with other agents, an agent must be able to translate its internal information about its goals, its perception of the world, and its current actions, into a form that can be understood by other agents. The converse is translating messages from other agents into an internal representation that the agent can work with. The general solution to both requires full natural language.

In the current version of TacAir-Soar, we finesse the general problem and use a template-based approach where we prespecify the form of the messages that the system can generate and accept. Our agents know when to generate these messages. They also know how to interpret these messages and modify their own internal knowledge structures appropriately. Thus, we've implemented the types of communication required by our agents but gone no further. However, the human interactions themselves are very stylized, with a strong emphasis on encoding information into short phrases whenever possible. For example, a pilot might send "bogey dope" to a ground controller, which is request for information on the current bogey that the pilot is engaging. This approach has been successful for the types of communication our agents need to produce, and is natural enough so that humans can fly as lead or wingman with our agents using a simulation interface with menu-driven communication that approximates the cockpit of an F-14, MiG-29, or E-2C (van Lent & Wray 1994). The human communicates with our agents through a menu-driven interface, and the messages from our agents are understandable to the human. However, this approach will break down when extended to unrestricted natural language interactions with real pilots. To that end, we are investigating general natural language approaches (Rubinoff & Lehman 1994).

Discussion

On the surface, our approach might appear to suffer from rigidity because it depends on a set of "canned" interactions based on existing doctrine and tactics. However, our agents are not blindly applying a fixed doctrine independent of changes in the environment. Instead, our agents are continually reassessing the situation, dynamically stringing together bits and pieces of existing doctrine and tactics that are appropriate to each situation, possibly generating novel behavior (when viewed over time). Thus, our agents do very well as long as the situation is covered by some combination of existing military practice (which includes defining new missions and many types of changes to the organizational structure). In completely novel situations, our agents will use whatever pieces of doctrine that are relevant to the situation. However, our agents do not have the ability to step back and reason from first principles about what would be a new, possibly novel coordinated response to the situation (although this is one of our research areas).

The long term goal of our work is to build intelligent autonomous agents. In this paper, we have demonstrated that it is possible to create agents for a complex environment in which coordination is critical. Our approach has been straightforward. We try to model the coordination methods used by humans, and to date, we have implemented coordination without negotiation, extensive internal agent modeling, or special architectural mechanisms. The coordination arises out of shared doctrine and tactics, shared knowledge of missions, observations of behavior, and explicit communication. Our success is heavily dependent on four characteristics of our domain which simplified the implementation of coordination: the shared goals of the agents, the expert-level performance (and knowledge) of the agents, the well-defined methods and procedures of the military that we are modeling, and the availability of experts that are willing and able to provide the details of procedures.

In the near future we will be extending our agents to all military air missions, including helicopters, joint mission with ground forces, and large scale coordinated strikes (involves 20-30 aircraft at once). These new missions with allow us to evaluate the sufficiency of our approach in an even more complex and "coordination-rich" domain.

References

Bond, A. H., and Gasser, L. 1988. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann.

Calder, R.; Smith, J.; Courtenmanche, A.; Mar, J.; and Ceranowicz, A. 1993. ModSAF behavior simulation and control. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation.*

Durfee, E. M., and Lesser, V. R. 1987. Predictability versus responsiveness: Coordinating problem solvers in dynamic domains. In *Proceedings of the tenth IJCAI*, 875–883. San Mateo, CA: Morgan-Kaufmann.

Durfee, E. M. 1988. *Coordination of Distributed Problem Solvers.* Boston, Mass.: Kluwer Academic Publishers.

Ephrati, E., and Rosenschein, J. S. 1992. Constrained intelligent action: Planning under the influence of a master agent. In *Proceedings of AAAI-92.* MIT Press.

Huffman, S. B.; Miller, C. S.; and Laird, J. E. 1993. Learning from instruction: A knowledge-level capability within a unified theory of cognition. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, 114–119.

Laird, J. E.; Jones, R. M.; and Nielsen, P. E. 1994. Coordinated behavior of computer generated forces in TacAir-Soar. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation.*

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33(3).

Rao, A.; Lucas, A.; Selvestrel, M.; and Murray, G. 1994. Agent-oriented architecture for air combat simulation. Technical report, The Australian Artificial Intelligence Institute. Techinal Note 42.

Rosenbloom, P. S.; Johnson, W. L.; Jones, R. M.; Koss, F.; Laird, J. E.; Lehman, J. F.; Rubinoff, R.; Schwamb, K. B.; and Tambe, M. 1994. Intelligent automated agents for tactical air simulation: A progress report. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, 69–78.

Rosenschein, J. S. 1993. Consenting agents: Negotiation mechanisms for multi-agent systems. In *IJCAI 93.*

Rubinoff, R., and Lehman, J. F. 1994. Natural language processing in an IFOR pilot. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation.*

Shoham, Y., and Tennenholtz, M. 1992. On the synthesis of useful social laws for artificial agents societies (preliminary report). In *Proceedings of AAAI-92.* Morgan Kaufmann.

Smith, R. G. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computing* 29(12):1104–1113.

steering committee, T. D. 1994. The DIS vision: A map to the future of distributed simulation. Technical Report IST-SP-94-01, Institue for simulation and training, University of Central Florida, Orlando, FL.

Sycara, K. 1989. Multiagent compromise via negotiation. In Gasser, L., and Huhns, M., eds., *Distributed Artificial Intelligence, Vol. II.* London: Pitman. 119–138.

van Lent, M., and Wray, R. 1994. A very low cost system for direct human control of simulated vehicles. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation.*

Natural Language Processing for IFORs: Comprehension and Generation in the Air Combat Domain

Jill Fain Lehman, Julie Van Dyke, and Robert Rubinoff
Carnegie Mellon University
Pittsburgh, PA 15213
jef@cs.cmu.edu

Abstract

In support of the Soar/IFOR project's goal of providing intelligent forces for distributed interactive simulation environments [Laird *et al.*, 1995], the NL-Soar project works toward the implementation of a full natural language capability for Air-IFOR agents. In this paper we discuss the design of that language capability (NL-Soar) and its integration into TacAir-Soar agents. In particular, we demonstrate how NL-Soar's linear complexity, interruptibility, and atomaticity of language processing provide language comprehension and generation processes that do not compromise agent reactivity.

1 Introduction

Autonomous intelligent forces (IFORs) play an increasingly critical role in both large-scale distributed simulations and small-scale, focused training exercises. An IFOR is a complex agent that requires diverse capabilities to perform at a useful level of functionality. Since an IFOR's role will often be to replace one or more individuals in an engagement, the ability to communicate in natural language can be a key capability contributing to its overall performance. An agent that is rigid in its communicative ability may introduce a brittleness into the simulation (i.e. a tendency to fail in unexpected ways) that has nothing to do with imperfections in strategic or tactical knowledge. Thus, in building TacAir-Soar agents to participate in beyond-visual-range combat [Laird *et al.*, 1995], an NL capability is needed to ensure reactive, human-like performance in basic interactions among pilot, wing, and air intercept control (AIC).

In [Rubinoff and Lehman, 1994a] we identified three main characteristics of communication during air combat that present challenging areas of research: (1) it occurs in real-time, (2) it must seamlessly integrate with the agent's non-linguistic capabilities, e.g. perception, planning, reasoning about the task, and (3) its content must be comprehended and generated in accordance with performance data, i.e. with all of the idiosyncratic constructions, ungrammaticalities, and self-corrections found in real language. Within the context of these research issues, we introduced NL-Soar, a language comprehension and generation capability designed to provide integrated, real-time natural language processing for systems built within the Soar architecture [Lewis, 1993; Nelson *et al.*, 1994a; Nelson *et al.*, 1994b; Rubinoff and Lehman, 1994b]. In this paper we concentrate on issues (1) and (2), exploring our progress toward their solution using NL-Soar in Soar-based Air-IFOR agents.

2 Demands of reactivity

The naive approach to communication between agents, and the one available using off-the-shelf technology, treats language as front-end and back-end interfaces. Messages are comprehended by a front-end module, which creates a system-dependent representation of the message that can be used by the other modules responsible for the agent's behavior. Similarly, when an agent needs to send a message, that same representation is passed to a back-end module that generates an output message to be directed to other agents.[1]

This makes language an all-or-nothing endeavor, the implications of which can be seen in Figure 1. In this typical tactical air scenario, blue is flying an intercept (1) and is actively pursuing the goal of achieving its launch acceptability region (LAR) when an incoming message arrives (2). The message is buffered until the current goal is achieved and blue has fired a missile (3). Next, processing of the input begins (4); it ends sometime after red has returned fire (5) and (6). Only after the communiation has been understood can blue begin its evasive maneuver (7).

It is clear that reactivity is compromised if understanding must be postponed until the current

[1]The approach being described here does not depend in any way on the content of the message or the style of language accepted and generated. Thus it would apply equally whether the language passed is natural language or a formal communication protocol (such as CCSIL [Salisbury, 1995]).

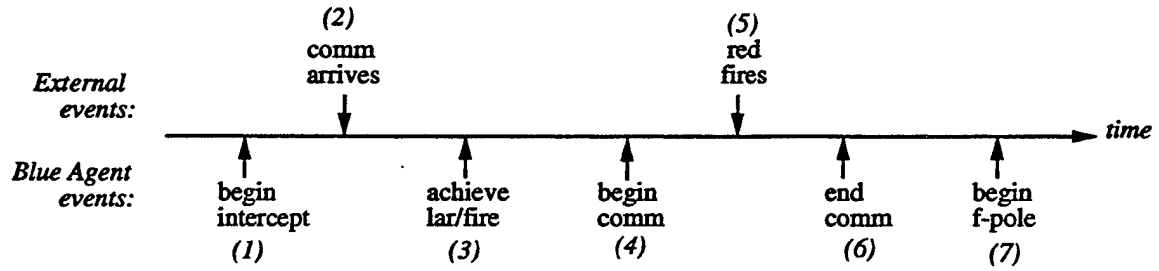


Figure 1: All-or-nothing: a communication model that compromises reactivity

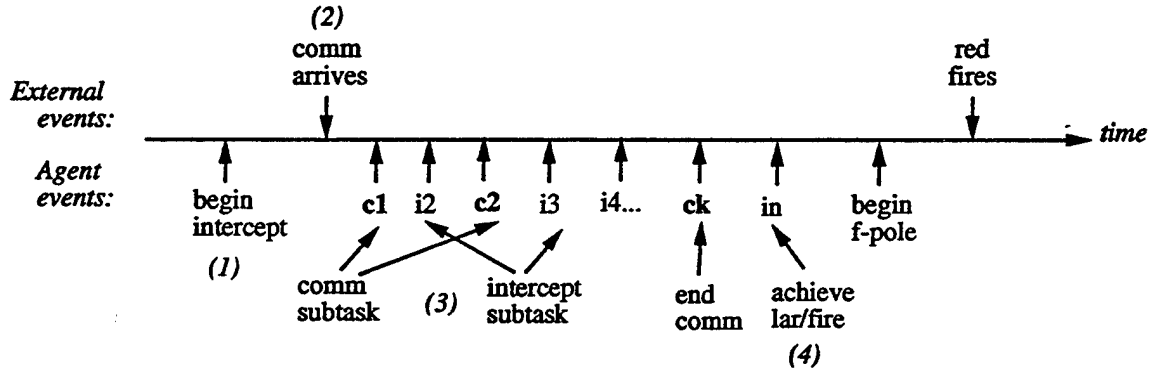


Figure 2: Reactive communication: interleaving comm and non-comm subtasks

goal has been accomplished, and then is pursued to the exclusion of all else. In particular, two cases are cause for concern. Consider first what happens at (2) if the content of the message is relevant to the situation at the time it is received. In this case, buffering the message leads, at best, to wasted processing in the future (when the message has become obsolete). At worst buffering compromises the decision making of the agent by precluding access to timely, necessary information. To remove this possibility, we could modify the control of the agent to always attend to communication needs first. But this would simply put us in the second problematic situation more often.

In this second case (4), if the content of the message is not critical, devoting processing to it rather than other things can compromise the agent's reactivity as well. In short, shutting out either communication processes or non-communication processes can be equally dangerous. The point, of course, is that you can't tell which situation you will be in until you process the message, at which time it is too late to change your mind.[2]

Figure 2 gives a more desirable version of the same task events. Again, the pilot is flying an intercept (1), trying to achieve firing position when a message arrives (2). The message is attended to immediately, its processing interleaved with the ongoing effort to achieve LAR (3). In this example, the message is completely processed by the time the pilot is in a position to fire (4), and evasive maneuvers can be started immediately, well before red returns fire.

The model in Figure 2 overcomes the problems in the simpler model of Figure 1 by intertwining the different strands of agent behavior at the sub-

[2]Dedicating a separate, parallel process to communication might ameliorate the problem but won't necessarily solve it. A separate process will be able to comprehend or generate the message while the agent is performing other tasks, but will have to work in isolation, i.e. cut off from the changing situation and goals of the agent. To the extent that there is relevant information that is unavailable during communication processing, the agent may formulate interpretations or communications that are inappropriate or out of sync. To the extent that the relevant information is communicated to the language process, parallelism is lost. In the tactical air domain information is updated quickly, and so an increasing proportion of CPU cycles will be necessary to keep the two processes in sync. Thus, to maximize reactivity, we conjecture that a separate process for communication would be more costly and no more effective than the method outlined in the following section.

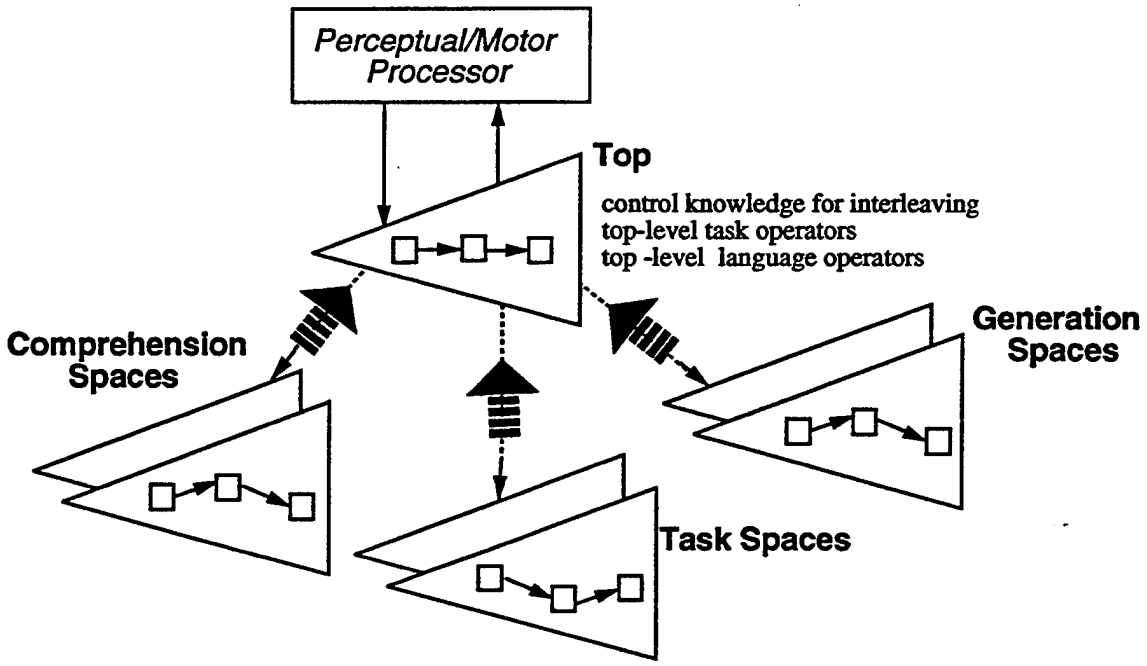


Figure 3: An Example of the Top-state Control Model

task level rather than at the full task level. In other words, we can view the all-or-nothing model as a degenerate case of Figure 2, one in which the granularity of the interleavable components is as large as possible. As we have seen, the disadvantage of choosing the maximal grain size is that the components are too large for the agent to behave in a timely fashion.

3 Achieving Interleavable Communication

For NL-Soar to provide a reactive, interleavable language capability for IFOR agents, the system as a whole must have three properties: *linear complexity*, *interruptability*, and *atomaticity*. The first property, linear complexity, means that processing to understand or generate a message must take time that is roughly linear in the size of the message. This is necessary to keep pace with human rates of language use. The second property, interruptability, ensures that time-critical task behaviors cannot be shut out by language processing (and vice versa). The third property, atomaticity, ensures that if language processing is interrupted, partially constructed representations are left in a consistent and resumable state.

To understand how NL-Soar provides the desired communication model, we must first briefly review the components out of which Soar systems are organized. Figure 3 is a graphical representation of a hypothetical Soar system that uses NL-Soar for comprehension and generation. Linguistic processes, like all processes in Soar, are cast as sequences of *operators* (small arrows) that transform *states* (boxes) until a goal state is achieved. The triangles in the picture represent *problem spaces* which are collections of operators and states.[3] The comprehension problem spaces contain operators that use input from the perceptual system to build syntactic and semantic structures on the state; the generation problem spaces contain operators that use semantic structures to produce syntactic structures and motor output. Note that the problem space labelled Top is the only space connected to the perceptual and motor systems and it is this space that is designated by the Soar architecture; all other problem spaces are provided by the system designer.

The dotted lines in the figure represent Soar impasses which arise automatically when there is a lack of knowledge available in the current problem space. When an impasse arises, processing continues in a subspace until the goal state in the subspace is reached. Note that impasses are a general recursive structure (a subspace can impasse into another subspace) that gives rise to a goal/subgoal hierarchy, or *goal stack*. The thick banded arrow

[3] For more details on how Soar uses problem spaces, states and operators to organize its processing see [Laird *et al.*, 1987; Laird *et al.*, 1995].

that overlays the impasse represents the resolution of the impasse, and the new knowledge (called *chunks*) that results from Soar's learning mechanism. Chunks capture the work done in the subspace, making it available in the superspace without impasse during future processing. This means that when a system structured as in Figure 3 is *fully chunked* all of its behavior will be produced by operators in the Top space.

We now have all the pieces to build an interleavable language capability. In the following sections, we address how to achieve linearity, interruptability, and atomaticity using these components. For the time being we will consider communication only in systems where the desired behavior shown in Figure 2 would occur completely within the Top problem space when fully chunked. We call a system organized in this way, a *Top-state control* model.[4]

3.1 Achieving Linear Complexity

Communication in an IFOR must occur in real-time to keep pace with the flow of human events. This is not a statement about how fast the system must run, per se. Rather, it is a theoretical statement about how processing must occur within the system. Although there is some variability (some words do reliably take longer to process than other words), in general, the amount of time taken by people is linear in the number of words in the utterance. A number of design constraints follow from this simple regularity [Lehman *et al.*, 1996], e.g. construction of the meaning of the sentence must proceed incrementally, and different knowledge sources (syntax, semantics, pragmatics) must be applied in an integrated rather than pipe-lined or multi-pass fashion. NL-Soar provides these properties [Lehman *et al.*, 1991a; Lewis, 1993]. Briefly, the system relies on Soar's notion of impasse to control the search through its linguistic knowledge sources, and then on Soar's learning mechanism to compile the disparate pieces of knowledge into an integrated form that can be applied directly (i.e. in approximately constant time/word) in the future.

Figure 4 depicts the process graphically for one type of language operator, expanding the left portion of Figure 3. Consider the arrival of a new word into the Top state and assume that the system has not encountered the word in a similar context in the past (i.e. the system has no pre-chunked knowledge about how to process this word). Once the word has been attended to, the learn-comprehension operator will be selected, after which an impasse will arise. Problem solving

[4] As we will see in Section 4, this is not the only structure permitted by Soar, but it is a valid organization and the simplest place to begin.

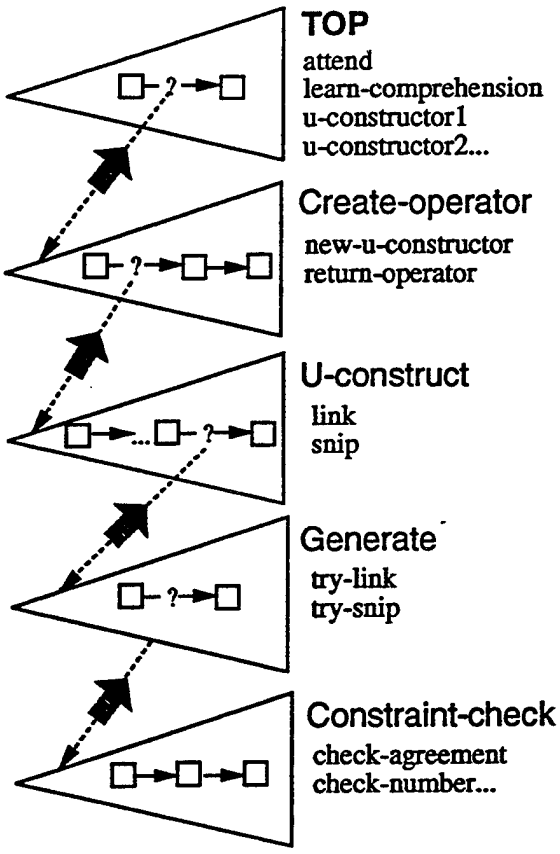


Figure 4: Achieving linearity through learning

will continue in the Create-operator space which will generate a symbol for a new *u-constructor*. A u-constructor is a language operator that fits the new word into the current syntactic structure for the message. The u-constructor is composed piecemeal in the U-construct space which performs links and snips on syntactic trees based on knowledge provided by Generate and Constraint-check. As the goal of each subspace is achieved, each impasse is resolved, creating chunks. Only two kinds of chunks concern us here. The implementation of the u-constructor is contained in the chunks created when the impasse between Create-operator and U-construct is resolved. This means that the syntactic tree that resulted from the sequential links and snips that were done in the lower spaces will now be produced immediately whenever this u-constructor executes. The u-constructor itself is returned from Create-operator to the Top space, resulting in a chunk that tells when this u-constructor can apply in the future. Note that the next time this word is seen in a similar contect, this chunk will propose the new u-constructor directly in the Top state. In other words, *once we have learned the top-level opera-*

tor, no impasse will occur. Instead, the (possibly lengthy) problem solving that took place in the subspaces has been compiled into a single Top-space operator that executes directly to build the relevant syntactic structure on the Top state.

Figure 4 shows how the application of general knowledge about syntax is contextualized and made efficient. A similar story can be told about *s-constructors*, the Top-space operators that fit the new word into the semantic and discourse structures maintained on the Top state. Thus, once behavior is fully chunked, the arrival of a message results in only a small number of Top operators per word, the linear complexity we were after. Equally important, the language process itself is now represented in the Top space in terms of more finely-grained operators (u- and s-constructors) that create the opportunity for interleavability. On the generation side, of course, there is a different task decomposition producing a different set of Top-space operators, but the principle is the same.

3.2 Achieving Interruptability

In Soar, agent behavior is produced by the application of operators to a state. Moreover, the architecture defines the application of an operator as a non-interruptable unit of work. In other words, once an operator has been selected for application, all the state changes associated with that operator are guaranteed to be made before any other operator is selected. What does this mean for NL-Soar? In short, it means that the Top-level language operators dictate the granularity of the interleavable components. To anchor the point in the context of Figure 4, once a u-constructor exists, we cannot interleave changes to the syntax tree with other non-linguistic tasks. Put more strongly, once the u-constructor is selected, all other subtasks are shut out for the duration of its application. In addition, if the Top state changes during the application of the u-constructor (via perception), those changes are effectively invisible until the u-constructor's state changes have been made.[5]

How is this situation different from the one in Figure 1, where lack of interruptibility meant reactivity was diminished to the point of inviting wasted work, if not disaster? The difference here is that the granularity of NL-Soar's operators is small enough to allow interruptibility below the full task level. The current scheme separates the work of attention from work done to the syntactic tree (u-constructors) from work done to the semantic and discourse models (s-constructors). Thus, the current comprehension capability allows for interruption between each set of state changes. Note, however, that we could have made this choice differently. We could, for example, build both syntactic and semantic structures in the impasse under the learn-comprehension operator. The resulting Top-space *comprehension operator* would effectively bundle all of comprehension into a single operator.[6] Alternatively, we could make link and snip the Top operators, giving an even finer grain. Although it is clear that the architecture permits a wide range of choices, choosing the right granularity is not a wholey unprincipled exercise. In general, the more work encompassed by a Top operator, the more specific will be the conditions under which it can apply. The more specific the conditions the less transfer of the knowledge to new situations and the more learning events will be required to get fully chunked language behavior. On the other side, the less work encompassed by a Top operator, the more operators per word there will be, until, eventually, the number will reflect some non-linear quantity (e.g. the size of the parse tree). In Section 4, below, we demonstrate how the operator granularity we have chosen allows both transfer and interleaving while maintaining linearity.

Now that we have language operators of a size that allows interruptibility, the next question that needs to be addressed is: how do you decide which type of operator, linguistic or non-linguistic, to select next? Many control schemes are possible, ranging from random selection to a complete partial ordering over all the operators in the system, to always attending to communication first (or last). In integrating NL-Soar with TacAir-Soar we will use random selection for its simplicity. What is important to remember, however, is that under Top-state control the selection decision is made on an operator by operator basis, not task by task.

3.3 Achieving Atomaticity

Recall that atomaticity ensures that if language processing is interrupted, partially constructed representations are left in a consistent and resumable state. Given our discussion above, it would seem that the architecturally enforced non-interruptability of operators would guarantee atomaticity as well. This is certainly true if all of the language behavior is impasse-free. Suppose, however, that the system is in the middle of learning a new u-constructor or s-constructor, as in Figure 4, when state changes create a preference for a non-linguistic Top-space operator. In this case,

[5] This is an overstatement. In fact, it is possible to encode knowledge in Soar in such a way that it is tied only to the state, not to any particular operator. Such knowledge will lead to state changes regardless of what operator is being applied. Since most task knowledge is tied to task operators, however, the discussion above is still a useful way to think about what's going on.

[6] An early version of NL-Soar did, in fact, use this scheme [Lehman *et al.*, 1991b].

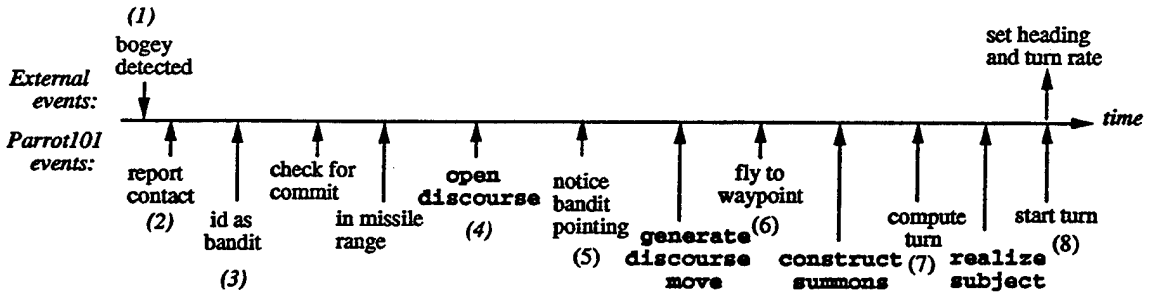


Figure 5: The lead TacAir agent composes a message while tracking a threat and flying

once the operator currently being applied in the lowest subspace is finished, the task operator will be selected in the Top space and the language goal stack will collapse. Can we be sure that we have been left in a consistent state so that language processing can be smoothly resumed?

The answer is yes because the design of NL-soar ensures that no changes are actually made to the language data structures on the Top state until the u-constructor is returned. Look again at Figure 4. The only operator that can result in changes to the Top state is Create-operator's return-operator. But if it is being applied when a preference is created for a Top-space task operator, then we know it will complete, the results will be returned, and the u-constructor proposal chunk will be built. If the subspace operator is not the return-operator, no results will be returned from the top-most impasse and no proposal chunk will be built for the u-constructor. Observe, however, that the conditions that led to the learn-comprehension operator in the Top space may well still obtain. So once the task operator has been applied, language may be resumed. Since no u-constructor was built, the system will have to rebuild the goal stack to continue. In practice, the situation is not as bad as it sounds because chunks may have been built in the subspaces during the previous learn-comprehension processing that were not specific to the particular u-constructor. These chunks will transfer to the current situation and the impasses that created them will be avoided.

4 Bringing it all together in TacAir-Soar

In Section 2 we argued that a communication capability for IFORs had to have three properties: linear complexity, interruptability, and atomaticity. In the previous section we introduced the Top-state control model in which whole tasks are interleaved on an operator-by-operator basis and communication is just another task. One of the interesting characteristics of systems organized as in Figure 3 is that the goal stack is never shared across linguistic and non-linguistic tasks; the need to understand or produce a message pulls the system out of a task goal stack. As a result, Top-state task operators, like the Top-state language operators, tend to represent subtasks of fairly short duration.

In contrast, systems like TacAir-Soar are composed of a Top task operator of very long duration, and a goal stack that reflects many levels of abstraction of that task. Each level stays active as long as it is being carried out. In particular, TacAir uses Soar's Top state to keep track of the "execute-mission" task, which stays active for the entire simulation. Under this will be a stack of sub-tasks, such as "mig-sweep", "intercept", "employ-weapons", and so on, each representing a more detailed view of what the agent is currently trying to do. Much of TacAir's knowledge of its current situation and goals is stored in sub-states associated with these subtasks, not on the Top state.[7] Thus, if TacAir switched to language in its Top state, it would lose much of this knowledge. Clearly, TacAir-Soar is incompatible with the Top-state control model outlined above. To understand how to modify Top-state control without sacrificing linearity, interruptibility and atomaticity, we must answer the question: what role, exactly, does the Top state play in maintaining each property?

For linear complexity, the role played by the Top state is simply a place to apply the so-called Top-state operators. In reality, what is critical for linear complexity is that there is an effective procedure for building the top-level language operators, and that only a small number of them are necessary for each word in the message. For interruptability and atomaticity, the Top state does play a more central role. Specifically, it must be the place where Top-level language operators leave

[7] A fuller description of TacAir-Soar can be found in [Laird *et al.*, 1995].

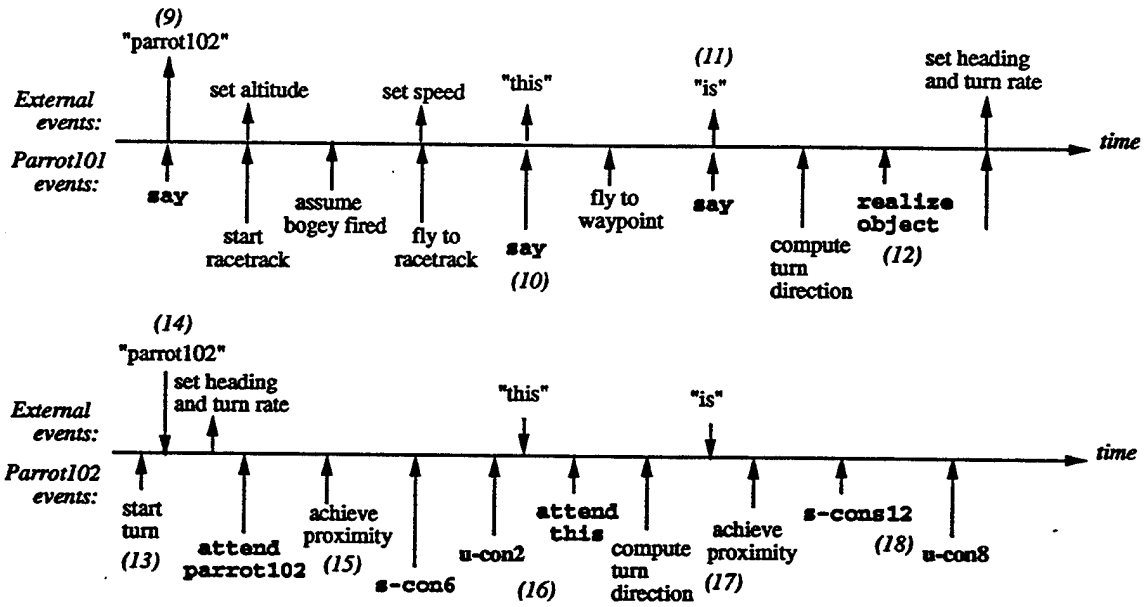


Figure 6: Figure 5 continued: Pilot continues to talk as wing begins to listen

their results because it is the only state that is guaranteed to still be in the goal stack when language processing resumes. Thus, *where* top-level language operators are applied is immaterial as long as they leave their results on the Top state where they can be found whenever, and wherever, language processing resumes.

Separating the question of where top-level language operators are applied from the question of where they leave their results allows us to define a variety of *virtual Top-state control* schemes. The simplest one, and the one we use when integrating NL-Soar with TacAir agents, is to interleave language operators with whatever task operators are available in the lowest problem space in the goal stack. Because the goal stack grows and shrinks over time, the interleaving of communication will take place more or less throughout the range of non-linguistic subtasks. The simplicity of the integration is extended by allowing the architecture to decide randomly between language and non-language operators whenever both types are applicable in the current situation.

Figures 5 through 7 capture a portion of the behavior of two TacAir-Soar agents running with a fully-chunked NL-Soar under virtual Top-state control.[8] In the scenario depicted, two pilots fly F14s as a section with a single red plane flying against them. Parrot101 is the lead and Parrot102 is the wing. The timelines in the figures show the operators that each agent executed in a particular engagement, together with those events in the external world that affect or depend on their behavior. Language operators are indicated via bold-face. For simplicity, the representation does not try to preserve the goal-subgoal relationship of the task operators.

In the time prior to the first event shown in Figure 5, the two planes have begun to fly in a racetrack configuration. The portion of behavior we are interested in begins when the lead notices the bogey (1), and must communicate the relevant information to its wing. The report-contact operator (2) posts a *communicative goal* on the Top state indicating that the agent wants to say something. Interleaving begins (somewhat unevenly due to the random control scheme) at (3). First, three task operators are executed in which the agent determines that the bogey is in fact a bandit, decides to check whether the commit criteria have been satisfied (they have not), and notices that the bandit is within missile range. Then, at (4), language operators begin to compose the message according to communication doctrine. The first step in any lead-wing communication is the

[8] Requiring NL-Soar to "learn while doing" would be equivalent to expecting the pilot to learn the domain language while flying the plane in battle. Consequently, we use off-line training to allow NL-Soar to learn from experience in a non-real-time setting. This gives the system the time it needs to integrate its disparate knowledge sources into the top-level operators discussed in Section 3.1. It is this highly compiled form of language knowledge that models an experienced pilot and provides real-time language behavior on-line.

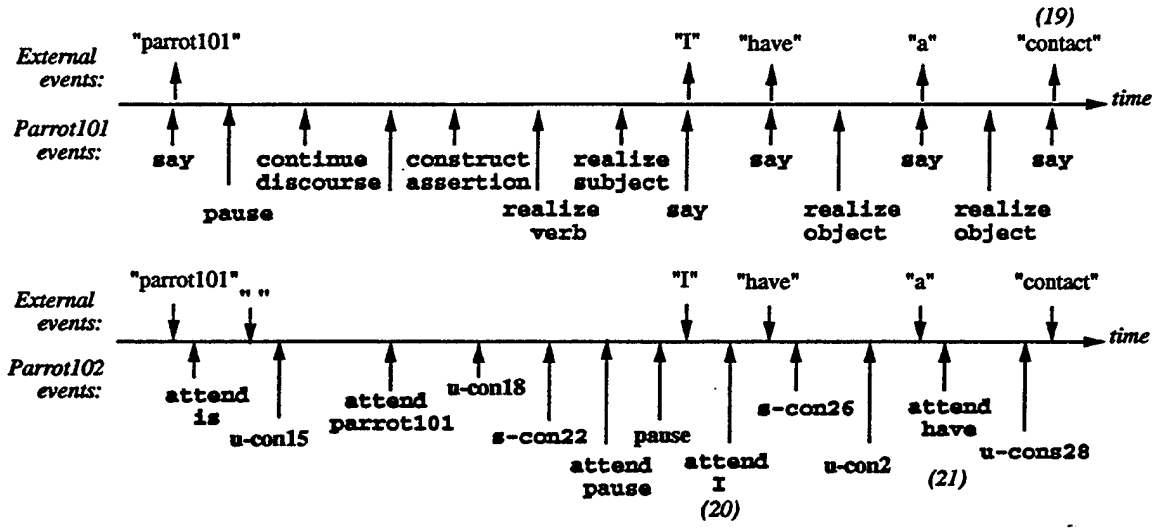


Figure 7: Figure 6 continued: completion of summons generation

exchange of callsigns, here, the sentence *Parrot102 this is Parrot101.* This is a domain-dependent instance of the more general class of utterances we call summons (for example the telephone exchange *John? It's Jill.*) The summons is constructed piece by piece using top-level generation operators (in boldface). Figure 5 shows this linguistic process interleaved with operators that contribute to situation awareness (5) and operators that fly the plane (6), (7), and (8).

Figure 6 continues the timeline for Parrot101 and introduces Parrot102 at the point just before the first word of the summons arrives into the agent's input buffer. The timelines are aligned by the linguistic output of Parrot101 and the linguistic input of Parrot102.

To this point in the scenario, the wing has simply been flying a racetrack with the lead. At (9) Parrot101 outputs the wing's callsign in the upper timeline. Note that this is done even though the construction of the remainder of the summons is still being interleaved with non-linguistic subtasks (10) through (12); both generation and comprehension are incremental. Meanwhile, shortly after Parrot102 has begun to turn (13), the callsign is heard (14). The lower timeline continues with comprehension of the first few words of the summons ((16) and (18)) interleaved with operators that keep the wing in formation ((15) and (17)). Note that the s- and u-constructors for the word *this* (18) fire after the word *is* has already been heard. This is partly because the lead's message is coming out quickly, and partly because the wing's attention has been focused on flying the plane. The input buffer that holds unattended speech has a decay rate; as in people, if speech goes unattended long enough (as it may if the pilot is in a stressful situation), it simply disappears from the buffer.

Figure 7 continues the interchange to the point that Parrot101 outputs the final word of the summons (19). There is no interleaving in this portion of the trace because both pilots are simply flying the long leg of the racetrack where no task operators are proposed. Notice that by the time the lead has begun the second portion of the summons, the wing has caught up on the comprehension side (19). The rapidity with which *I have a contact* emerges, however, once again results in buffered input for Parrot102 (21). Thus, linguistic processing continues in the wing agent after the lead has already begun to wait for a reply (not shown). As a final observation, note that the same u-constructor that processes *Parrot101* in Figure 6 also processes *I* in Figure 7 (u-constructor2). This is an example of where the granularity of the top-level operators affords some transfer of syntactic processing despite the difference in semantics (s-constructor6 vs. s-construct26).

5 Conclusions

The ability to communicate in natural language can be a key capability contributing to an IFOR's performance in both simulation and training exercises. In this paper we have discussed how the design of NL-Soar uses linear complexity, interruptibility, and atomaticity of language processing to provide a language capability that does not compromise reactivity. What we have not discussed, however, is the third area of interest identified in

[Rubinoff and Lehman, 1994a]: performance in accordance with empirical data from pilots in real-life simulations. Our continued work, therefore, will focus on making the NL-Soar integration more robust, including handling linguistic constructions specific to the domain and allowing for the interruptions and self-corrections that necessarily come with real language use.

6 Acknowledgement

This research was supported under subcontract to Carnegie Mellon University from the University of Michigan, as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Research Laboratory (NRL). The authors would like to thank BMH Associates, Inc. for their technical assistance, and gratefully acknowledge the system-building support of Greg Nelson.

References

[Laird *et al.*, 1987] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[Laird *et al.*, 1995] John E. Laird, W. Lewis Johnson, Randolph M. Jones, Frank Koss, Jill F. Lehman, Paul E. Nielsen, Paul S. Rosenbloom, Robert Rubinoff, Karl Schwamb, Milind Tambe, Julie Van Dyke, Michael van Lent, and III Robert E. Wray. Simulated intelligent forces for air: The soar/ifor project 1995. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, 1995.

[Lehman *et al.*, 1991a] J. Fain Lehman, R. Lewis, and A. Newell. Integrating knowledge sources in language comprehension. In *Proceedings of the Thirteenth Annual Conferences of the Cognitive Science Society*, 1991.

[Lehman *et al.*, 1991b] J. Fain Lehman, R. Lewis, and A. Newell. Natural language comprehension in soar: Spring 1991. Technical report, School of Computer Science, Carnegie Mellon University, CMU-CS-91-117, 1991.

[Lehman *et al.*, 1996] J. Fain Lehman, R. Lewis, and A. Newell. NL-Soar: Architectural influences on language comprehension. In *Cognitive Architecture*. Ablex Press, 1996. in press.

[Lewis, 1993] R. L. Lewis. *An Architecturally-based Theory of Human Sentence Comprehension.* PhD thesis, Carnegie Mellon University, 1993.

[Nelson *et al.*, 1994a] G. Nelson, J. F. Lehman, and B. E. John. Experiences in interruptible language processing. In *Proceedings of the 1994 AAAI Spring Symposium on Active NLP*, 1994.

[Nelson *et al.*, 1994b] G. Nelson, J. F. Lehman, and B. E. John. Integrating cognitive capabilities in a real-time task. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, 1994.

[Rubinoff and Lehman, 1994a] R. Rubinoff and J. F. Lehman. Natural language processing in an ifor pilot. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, pages 97–104, 1994.

[Rubinoff and Lehman, 1994b] R. Rubinoff and J. F. Lehman. Real-time natural language generation in nl-soar. In *Proceedings of the Seventh International Workshop on Natural Language Generation*, 1994.

[Salisbury, 1995] M. Salisbury. Command and Control Simulation Interface Language (ccsil): Status update. In *Proceedings of the the 12th Distributed Interactive Simulation Workshop*, 1995. Sponsored by STRICOM and the Institute for Simulation and Training (IST) at the University of Central Florida.

7 Biographies

Jill Fain Lehman is a research computer scientist in Carnegie Mellon's School of Computer Science. She received her B.S. from Yale in 1981, and her M.S. and Ph.D. from Carnegie Mellon in 1987 and 1989, respectively. Her research interests span the area of natural language processing: comprehension and generation, models of linguistic performance, and machine learning techniques for language acquisition. Her main project is NL-Soar, the natural language effort within the Soar project.

Robert Rubinoff is a postdoctoral research fellow in Carnegie Mellon's School of Computer Science. He received his B.A., M.S.E., and Ph.D. from the University of Pennsylvania in 1982, 1986, and 1992, respectively; his dissertation research was on "Negotiation, Feedback, and Perspective within Natural Language Generation". His research interests include natural language processing, knowledge representation, and reasoning. He is currently working on natural language generation within the Soar project.

Julie Van Dyke is a Research Programmer working on language comprehension in NL-Soar. She is also working toward an MS in Computational Linguistics with a focus on modeling language acquisition.

Intelligent Computer Generated Forces for Command and Control

Paul E. Nielsen
Department of Electrical Engineering and Computer Science
University of Michigan
1101 Beal Ave., Ann Arbor, MI 48109-2110
nielsen@eecs.umich.edu

> The clever combatant looks to the effect of combined energy, and does not require too much from individuals.
>
> Sun Tzu *The Art of War*

1. Abstract

The effectiveness of intelligent computer generated forces is limited by their ability to closely coordinate their actions within the overall battlefield situation. We have developed intelligent command and control agents which monitor large sections of the battlefield and deploy other forces for increased effectiveness. These agents have been demonstrated in the air to air, close air support, and air strike domains.

2. Introduction

Our goal is the development of intelligent forces (IFOR's), computer agents which are functionally indistinguishable from human agents in their ability to interact with the synthetic environment. The Soar/IFOR consortium, involving the University of Michigan, Information Sciences Institute of the University of Southern California, and Carnegie Melon University, is developing IFORs for all military air missions: air to air, air to ground, air supply, anti-armor attack, etc. IFORs must have many capabilities to be successful including: extensive knowledge, real-time reactivity, goal-directed problem solving, and planning. Additionally, they must coordinate their activities with other friendly forces (Laird *et al.*, 1995a).

To fully support very large scale battle field simulations, such as those envisioned for STOW-97, intelligent computer generated forces cannot act independently; but rather, they must coordinate their efforts for increased effect just as humans do. This requires a means and a method for coordination, the ability to convey coordination information, and the ability for large scale situation assessment. In military parlance this is commonly referred to as command, control, communications, and intelligence (C^3I).

This paper discusses our current state of development of intelligent, realistic C^3I agents for simulation in the air domain. These agents have been implemented using ModSAF (Calder *et al.*, 1993) and the Soar/ModSAF interface (Schwamb *et al.*, 1994).

The remainder of this introductory section provides an overview of the C^3I domain and some motivation for this work. Section 3 has a description of the C^3I agents implemented by this project to date. Section 4 discusses the general responsibilities of each agent and goes on to show how our agents demonstrate each of the C^3I functions. Section 5 provides an extended example of the interaction between multiple C^3I agents and a section of planes flying close air support. Section 6 discusses research and open problems. Finally, section 7 provides general discussion and conclusions.

2.1. Domain Overview

Previous work in computer generated forces has either focused on individual agents working in relative isolation or groups of agents which may be treated as a whole (Rao *et al.*, 1994). A notable exception is (Ballas *et al.*, in press). These approaches avoid the problems of C^3I by allowing human guidance, but when the agents number in the tens of thousands, finding enough people to control them is infeasible.

In 1994, the Soar/IFOR project was tasked to provide automated pilots for all air vehicles and mis-

sions in support of STOW-97. (See (Laird *et al.*, 1995b) for an overview of the current state of this project.) In order to accomplish this task we needed to extend the scope of the project to include those interactions necessary between pilots and controllers, even if they are not airborne. For example, orange agents are at a severe disadvantage if they cannot rely on ground based radar control (GCI) to track threats outside the limited scope of their own radar.

The most C^{3}I intensive missions we have implemented to date are air to air combat and close air support (CAS). In the air to air domain, the controller may be responsible for maintaining a defensive perimeter around the carrier battle group, locating potential threats, confirming that an unknown aircraft is a threat, providing timely updates until friendly planes have radar contact, then issuing additional information in response to queries.

While air to air combat has a single (or small number of) controllers, in contrast, the close air support domain demonstrates a wide variety of controllers. In the CAS domain, the attack planes must have detailed integration with multiple agents because of close proximity between targets and friendly forces. These controllers communicate with the planes (locating targets and deconflicting) as well as amongst themselves (requesting missions and allocating forces.)

2.2. Motivation

The primary motivation for doing this work is to develop realistic C^{3}I agents. The IFOR C^{3}I agents should be indistinguishable from human agents performing similar functions. This involves believable interacts with the simulator as well as interactions with other agents and humans at a natural level. By basing IFOR agents on Soar, a theory of cognition (Laird & Rosenbloom, 1994; Laird *et al.*, 1987; Newell, 1987), and modeling not only the externally observable behavior, but plausible thought processes which are necessary to produce realistic behavior, we intend to overcome both dumb, canned responses and implausible, superhuman responses.

The second motivation for doing this work is effectiveness. Without C^{3}I agents our automated pilots have only limited ability to sense and interact with their environment. Enemy agents can sneak up behind them or fly around them. In addition the automated pilots have only limited ability to change their mission. Without the large scale perspective provided by the controller, they don't even realize that there might be a need to change their mission.

Adding C^{3}I can increase the level and types of applications for military simulation. As battlefield simulators become more realistic, we want to make them available for more advanced purposes. The major use of air simulators to date is in pilot training. By providing intermediate level controllers, we expect to make simulation usable not only in pilot training, but also in training human controllers to interact with and control these controllers.

Finally, we wish to study human cognition and the ability to model it in Soar. C^{3}I provides a new domain for this research which suggests more knowledge and exhibits different types of knowledge than that used by aircraft pilots.

3. C^{3}I Agents

In order to increase realism and promote playability at various levels, we base C^{3}I on existing techniques currently in use by military organizations and embody them in specialized agents corresponding to military controllers. Thus there is a direct one to one mapping between our agents and humans.

Currently, we have operational versions of the following C^{3}I agents:

- Air Intercept Controller (AIC) which assigns planes to stations, spots threats, and provides information about enemy planes. The AIC is airborne, situated in a plane with a large radar, such as an E-2C.
- Ground Controlled Intercept (GCI) performs the same sort of mission as an AIC but is ground based and immovable.
- Forward Air Controller (FAC) which locates targets and provides final directions for close air support. Forward air controllers may be either ground based or airborne (FAC(A)).
- Direct Air Support Center (DASC) which assigns aircraft to missions, potentially alters the missions, and hands off attack missions

to the FAC. The DASC is ship based, usually on the aircraft carrier.

- Tactical Air Direction Controller (TAD) directs air operations within the Amphibious Operations Area (AOA) prior to the establishment of a DASC. The TAD is also ship based and may be co-located with the DASC.
- Fire Support Coordination Center (FSCC) determines the type of support to utilize (CAS, artillery, naval gunfire). If CAS is determined it generates a Joint Tactical Airstrike Request and coordinates CAS requests with the DASC. The FSCC is ground based within the AOA.
- Tactical Air Command Center (TACC) which provides air traffic control, routing, and deconfliction within the AOA. The TACC is ground based and usually co-located with the FSCC.

In the following section we explore how agents demonstrate the capabilities necessary for coordinating the behaviors of multiple agents.

4. Responsibilities

In addition to the specific responsibilities of each agent given above there are several general responsibilities associated with C^{3}I agents. These responsibilities are broken out into separate topics, but it must be realized that to work effectively all of these activities must be going on simultaneously.

4.1. Command

C^{3}I agents are responsible for mission initiation as well as tracking and modifying the mission as it develops. Typically the planes will have a prebriefed mission, but often this mission will need to be changed or replaced entirely as the battlefield situation developed. Our command agents can change almost every aspect of a mission including assignment of individual CAP[1] stations, routes, target times, and the final targets.

In order to effectively carry out their command function, C^{3}I agents need to have a command organization. We've observed two different command organizations for C^{3}I agents.

[1] Combat Air Patrol

In the air to air domain command is centralized. Either the AIC or the GCI are responsible for all air traffic. These agents provide continuous control and information for many sections of planes. Though there may be multiple controllers acting at the same time they have clearly separated duties, and there is very little interaction.

In contrast, in the CAS domain command is decentralized. As the planes fly through different regions they are directed by multiple controllers, all of which are responsible for the ultimate success of the mission. Though there is still a chain of command, because of limited numbers of radios and limited broadcast range the planes may not be in continuous contact with any single controller.

The controllers in CAS need to coordinate not only the planes, but also themselves. The TACC, DASC, FSCC, and FAC have to form a distributed control network in which mission requests and assignments are propagated through the network.

4.2. Control

The mission of a controller is to continually assess the situation then allocate, or re-allocate, forces for maximum effect.

The combined knowledge of overall mission objectives and threat detection makes controllers uniquely capable of resource allocation. They need to assess the resources available and when future resources might become available, balanced against current and potential threats. They must synchronize their own forces, and their efforts with respect to other controllers. Higher level controllers have to trade off the utility of multiple potential assignments for maximal effectiveness, while low level controllers can only shout louder hoping to increase the priority of their request for resource allocation.

Poorly coordinated attacks can be weak and ineffective. One way C^{3}I agents coordinate is by synchronizing attacks through timing constraints. For example, in the CAS domain, when bombing in tight proximity to friendly troops, timings must be accurate to plus or minus ten seconds to avoid interference with friendly troops.

To accomplish this C^{3}I agents must be capable of real-time reactive planning. Both threats, friendly forces, and messages from other controllers may

arrive at any time. The overall battle plan must be incrementally supplemented with new information so that we seize opportunities and knowingly avoid or confront risks.

Soar provides several capabilities which help manage these real-time asynchronous inputs. First, the decision of what to do next is handled through production rules. During each decision cycle all relevant rules are tested and allowed to fire in parallel. Thus the sequence of execution is not fixed.

The real-time is requirement handled by making the speed of operator execution comparable to experimental results in humans (Newell, 1990). Since this can only guarantee soft real-time, our agents will react quickly, but may fail to react quickly enough when faced with overly complex situations, just as people do. Limiting the number of available choices increases the speed of decision making. Soar uses operator subgoaling to provide a context for focusing decisions on information relevant to the current situation. For example, when under attack and bugging out an E-2 might not be overly concerned with planning the course to its CAP station.

Another way to increase military effectiveness is to decrease the interference from one's own forces. In actual combat (as opposed to simulation) this will have serious morale consequences. The deconfliction duties we've implemented range from air traffic control to route planning to explicitly informing the plane of the location friendly forces.

4.3. Communication

The nature of communication is that commands must be brief, and commands must be clear. C^{3}I agents must communicate relevant information in a timely and effective manner. Communication can range from simple (e.g., "proceed as briefed" or "negative") to very complex, such as a nine line brief shown in figure 1.

The domain of military communication is well researched, and the military jargon provides a form of communication which is brief yet maximizes the communication of necessary knowledge without undue overhead. We attempt to model C^{3}I using standardized forms, realistic dialog from actual communications of former pilots, and examples from training manuals whenever possible. We believe that by making communication explicit and

PREPLANNED, ON-CALL CAS
NINE/TWELVE LINE BRIEF

MISSION # 20-038 ORDN 2 X MK 83 LGB
ROUTE ELMER-PANTHER ENR FREQ BLUE
MISSION CODES:
CONTINUE SHARK CHANGE SNAPPER
CANCEL MARLIN ABORT FLOUNDER
CP: DODGE CALL PYTHON ON BLACK
1. IP WANDA
2. HDG 112 MAG L / R
3. DIST 8.5 NM
4. TGT ELEV 450 MSL
5. TGT DESC TROOP CONVOY ON BRIDGE
6. TGT LOCATION YH 205 500
7. MARK: TYPE LASER CODE 1688
8. FRIENDLIES NW 5000 M
9. EGRESS SOUTH TO FORD, THEN COUGAR
10. BCN-TGT BRG ____ MAG / BCN GRID
11. BCN-TGT DIST ____ METERS / TGT GRID
12. BCN ELEV ____ FEET
TOT / TOT 6400 (MIN / SEC) BACK
REMARKS / BDA ____

Figure 1: Nine/twelve line brief

based on human communication we can offer an approach to better human interaction and easier evaluation of the results of a simulation.

The approach used by the military, and the approach we've adopted, is to use a shared format for all communication. Complex commands use a standard template to reduce transmission time and ensure all relevant information has been communicated.

To compensate for lost messages and electronic interference we repeat messages until confirmation is forthcoming . The receipt of commands must be confirmed through "roger," or if some action is necessary, by the recipient either "wilco" (will comply) or "negative" (will not comply).

While we have yet to incorporate a general natural language understanding system with TacAir-Soar, the commands used are based on the actual English communications used between controllers and pilots in similar situations. This makes it easier to understand the behavior of the IFOR commanders, and allows human communication with the IFOR commanders. In order to communicate with other CGFs we will be adopting CCSIL pro-

tocols (Salisbury, 1995).

4.4. Intelligence

The most important responsibility of an air controller is to locate, identify, and track threats. "Timely interception is totally dependent of two factors: early detection and positive identification"(Gunston & Spick, 1983). The need to track the threat arises because enemy agents are eminently uncooperative. Some early failures of our fighter agents acting alone arose because human pilots would feign an attack from one direction, then beam or drop and attack from a different direction. The more powerful radar capabilities of the AIC and GCI makes our agents less vulnerable to these tactics.

Each agent has limited capability. Controllers are limited by weapons,[2] maneuverability, and speed when compared with the targets they must defend against. To compensate for this lack of ability they provide greater situational awareness either through proximity (e.g., a FAC) or superior equipment (such as an E-2's radar). They must use this awareness to perform continuous intelligence gathering. Without this information even a veteran pilot may be defeated by a poorly equipped pilot of lesser training.

5. Example scenario

Figures 2 through 8 illustrate some of the interaction between command agents and combat aircraft during a close-air support mission. All of this dialog is taken from a simulation run of a close air support mission.

Our agents include a section of F-14d fighters (lead by Falcon14), a TACC (Icepack), an FSCC (Bronco), a DASC (Mustang) and a FAC (Rattler). Each utterance is preceded by the name of the speaker and the radio frequency used for this communication. The frequencies are color coded to match the encryption scheme used in the communication.

In figure 2 the two planes check into the amphibious operations area (AOA) with Icepack. The exact form of the plane's initial check-in message is specified in the SPINs (SPecial INstructions) and may vary across scenarios, but will convey the essential information 1) who I am, 2) where I am, and 3) what am I doing here.

[2]Though, at least one E-2 pilot considers every friendly plane in the sky his weapon.

```
Falcon14 (white): Icepack this-is Falcon14
Icepack (white): go-ahead
Falcon14 (white): Falcon14
Falcon14 (white): mission-number 20-059
Falcon14 (white): proceeding-to Elmer
Falcon14 (white): angels 32
Falcon14 (white): time-on-station 1+30
Falcon14 (white): checking-in-as fragged
Icepack (white): roger
Icepack (white): Falcon14
Icepack (white): radar-contact
Icepack (white): cleared-to-enter-aoa
Icepack (white): proceed-as-briefed
Icepack (white): maintain angels 32
Icepack (white): check-in-with Mustang
Icepack (white): on orange
Icepack (white): at Tiger
Falcon14 (white): wilco
```

Figure 2: Mission checks in to AOA

Icepack recognizes this message and realizes that they are both friendly and supposed to be there. Icepack locates their corresponding blip on radar, gives them permission to enter the AOA, and does not change their mission.

Our TACC is capable of some low level air traffic control. In this case it consists of assigning unique, even altitudes to inbound flights, while outbound flights are expected to maintain odd altitudes.

Finally, Icepack hands off control to the next agent, Mustang, at a pre-briefed radio setting.

```
Rattler (silver): Bronco this-is Rattler
Rattler (silver): immediate-mission
Rattler (silver): target-is tank
Rattler (silver): target-location-is
Rattler (silver): x 127000
Rattler (silver): y 27500
Rattler (silver): target-time ASAP
Rattler (silver): desired-results destroy
Rattler (silver): final-control FAC Rattler
Rattler (silver): on green
Bronco (silver): roger Rattler
```

Figure 3: FAC sends tactical air request to FSCC

In figure 3 Rattler finds itself in the line of unfriendly fire and radios back to the FSCC that it needs support immediately. In addition it provides information sufficient for the FSCC to initiate a Joint Tactical Airstrike Request (JTAR).[3]

The JTAR includes target type, location, time, and desired results. Note that Rattler has elected to be the forward air controller for the mission and direct the final bombing run. The FSCC supplements this information with coordination and mission data.

```
Bronco (orange): Mustang this-is Bronco
Bronco (orange): request-number 28-59
Bronco (orange): immediate-mission
Bronco (orange): target-is tank
Bronco (orange): target-location-is
Bronco (orange): x 127000
Bronco (orange): y 27500
Bronco (orange): target-time ASAP
Bronco (orange): desired-results destroy
Bronco (orange): final-control FAC Rattler
Bronco (orange): on green
Mustang (orange): roger
```

Figure 4: FSCC radios DASC

In figure 4 Bronco (the FSCC) has determined that close air support is the logical response, and transmits the necessary information from the JTAR to Mustang (the DASC). If this were more realistic, the request would be transmitted in hard copy form rather than over the radio, but we are constrained with the information exchanges allowable through ModSAF.

In figure 5 the lead plane is approaching a holding point and checks in with Mustang. The plane's check in sequence has the same form as seen in figure 2.

At this stage Mustang alters the mission from its pre-specified course. Even though the planes have a pre-briefed mission, Mustang determines that the new mission is more important and redirects the flight to a new contact point (Chevy) and a new controller (Rattler) for further details.

[3]We've elected not to include an example of a Joint Tactical Airstrike Request because of its detailed nature. The nine/twelve line brief of figure 1 accounts for less than one sixth of its content by size.

```
Falcon14 (orange): this-is Falcon14
Mustang (orange): go-ahead
Falcon14 (orange): Falcon14
Falcon14 (orange): mission-number 20-059
Falcon14 (orange): proceeding-to Tiger
Falcon14 (orange): angels 32
Falcon14 (orange): time-on-station 1+30
Falcon14 (orange): checking-in-as fragged
Mustang (orange): Falcon14 this-is Mustang
Mustang (orange): proceed-as-briefed
Mustang (orange): check-in-with Rattler
Mustang (orange): on green at Chevy
Falcon14 (orange): wilco
```

Figure 5: Mission checks in with DASC

```
Mustang (green): Rattler this-is Mustang
Rattler (green): go-ahead
Mustang (green): expect-cas-mission 20-059
Mustang (green): call-sign Falcon14
Mustang (green): at Chevy
Rattler (green): roger
```

Figure 6: DASC contacts FAC

Figure 6 shows Mustang informing Rattler that help is on the way, who they are, and where to expect them. Rattler has no radar and will assume a plane approaching from that direction is the expected mission.

In figure 7 the planes finally arrive at the contact point for Rattler and check in according to the format seen in figure 2.

Figure 8 shows Rattler delivering a nine line brief similar to that shown in figure 1. This is an information intensive message which relies on the controller and pilot sharing a common communication model. All and only the necessary values are given sequentially without reference to meaning or line numbers.

What's being expressed here is that the initial point will be Joyce. The heading, in magnetic degrees, from the initial point to the target is 052. The distance from the initial point to the target is 18.6 nautical miles. The target's elevation is 0 above mean sea level. The target's description is a "tank". The target's coordinates are 127000 by 27500 in the X/Y coordinate system of ModSAF.

```
Falcon14 (green): Rattler this-is Falcon14
Rattler (green): go-ahead
Falcon14 (green): Falcon14
Falcon14 (green): mission-number 20-059
Falcon14 (green): 2 F-14d
Falcon14 (green): holding-at Chevy
Falcon14 (green): angels 32
Falcon14 (green): 10 MK82
Falcon14 (green): time-on-station 1+30
Falcon14 (green): no-laser-capability
Rattler (green): roger
Rattler (green): Falcon14
```

Figure 7: Mission check in with FAC

```
Rattler (green): standing-by
Rattler (green): with-9-line-brief
Falcon14 (green): ready-to-copy
Rattler (green): Joyce
Rattler (green): 052
Rattler (green): 18.6
Rattler (green): 0
Rattler (green): tank
Rattler (green): x 127000 y 27500
Rattler (green): wp
Rattler (green): sw 8000 meters
Rattler (green): Ford
Rattler (green): tot ASAP
Falcon14 (green): ASAP
```

Figure 8: FAC gives 9 line brief

The target will be marked with white phosphor.[4] There are friendlies in the area which are 8000 meters to the south-west. After the attack the plane should egress through Ford. And the attack should commence as soon as possible.

Falcon14 signals that he copies all of that information and agrees to it by repeating the time.

Following this, there are brief exchanges when the planes are spotted, cleared to drop, and for damage assessment.

6. Research Issues in C^{3}I

The development of C^{3}I agents presents several interesting research issues.

From a broader artificial intelligence perspective, C^{3}I presents interesting problems in reactive planning and managing dynamically changing goals in the face of uncertainty. The battle field environment is constantly changing. This requires a fast and efficient architecture to keep up with the speed requirements of the situation as well as a flexible architecture for incremental reasoning and reactive planning.

Most of the planning currently done by our system is reactive planning. In some situations the C^{3}I agents may have some time for decision making and should use this time for more deliberate planning. Recent research explores the possibility of incorporating planning and means-ends analysis mechanisms with our agents (van Lent, 1995; Wray, 1995).

This work is very closely related to distributed artificial intelligence. Since we are basing our work on an existing model which seems to work reasonably well, we can avoid many of the problems of distributed artificial intelligence systems. For example, our agents need not carry out protracted negotiations.

We've demonstrated that a template driven approach to language understanding provides a sufficiently flexible command language for many aspects of communication, but it's not clear how far this approach can be extended. More work needs to be done on natural language understand both for agent flexibility and ease of use in human computer interaction See (Lehman *et al.*, 1995) for recent work.

Though these agents were prepared to take part in the STOW-E demonstration, during rehearsal they were unable to handle the large number of other agents they saw in the world and crashed. This turned out to be a buffer overflow problem, but suggested several methods for reorganizing the way of IFOR agents handle large numbers of inputs. Currently, these IFOR agents will slow down and their performance will degrade as the number of other agents they have to consider increases.

In the immediate future we will address more mundane, but no less critical tasks of tracking fuel states and allocating fuel assets.

[4] The capability for marking a target does not yet exist.

7. Discussion

We have described the current state of development of C^3I agents used by Soar/IFOR. We have shown how the agents currently implemented demonstrate the specific aspects of the C^3I domain. Finally, we worked through an example which showed multiple control agents interacting with planes on a close air support mission.

We have demonstrated an ability to cope with incomplete knowledge and incrementally supplement information as it becomes available. This requires continuous situation assessment: commands, threats, and resources may arrive at any time.

We believe that automation must be pushed up the command hierarchy. As the number of simulated agents grows, people will have to supervise larger numbers of agents. We believe that the best way to do this is to emulate the present military command hierarchy. This has the advantage of ease of use (nothing new to learn), effectiveness (it has been proven through centuries of warfare), and ease of understanding.

8. Acknowledgments

This work was done in close cooperation with John E. Laird and Randolph M. Jones.

Thanks to BMH Associates, Inc. for their technical assistance, especially Craig Petersen, Mark Checchio, Tom Brandt, and Bob Richards.

This research was supported at the University of Michigan as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Research Laboratory.

9. References

Ballas, J. A. abd McFarlane, D. C., Achille, L. B., Stroup, J. L., Heithecker, C. H., & Kushnier, S. D. in press. *Interfaces for intelligent control systems.* Tech. rept. NRL Technical Report. Washington, D. C: Naval Research Laboratory.

Calder, R., Smith, J., Courtenmanche, A., Mar, J., & Ceranowicz, A. 1993. ModSAF behavior simulation and control. *In: Proceedings of the third conference on computer generated forces and behavioral representation.*

Gunston, B., & Spick, M. 1983. *Modern air combat.* New York: Crescent Books.

Laird, J. E., & Rosenbloom, P. S. 1994. *The evolution of the Soar cognitive architecture.* Tech. rept. Computer Science and Engineering, University of Michigan. To appear in Mind Matters, T. Mitchell Editor, 1995.

Laird, J. E., Newell, A., & Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial intelligence*, **33**(3).

Laird, J. E., Jones, R. M., & Nielsen, P. E. 1995a. *Multiagent coordination in distributed interactive battlefield simulations.* Tech. rept. Computer Science and Engineering, University of Michigan.

Laird, John E., Johnson, W. Lewis, Jones, Randolph M., Koss, Frank, Lehman, Jill F., Nielsen, Paul E., Rosenbloom, Paul S., Rubinoff, Robert, Schwamb, Karl, Tambe, Milind, Dyke, Julie Van, van Lent, Michael, & Wray, Robert. 1995b (May). Simulated intelligent forces for air: The Soar/IFOR Project 1995. *In: Proceedings of the fifth conference on computer generated forces and behavioral representation.*

Lehman, J. F., Rubinoff, R., & Van Dyke, J. 1995 (May). Natural language processing for IFORs: Comprehension and generation in the air combat domain. *In: Proceedings of the fifth conference on computer generated forces and behavioral representation.*

Newell, A. 1987. *Unified theories of cognition: 1987 William James lectures.* Available on videocassette from Harvard Psychology Department.

Newell, A. 1990. *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Rao, A., Lucas, A., Selvestrel, M., & Murray, G. 1994. *Agent-oriented architecture for air combat simulation.* Tech. rept. The Australian Artificial Intelligence Institute. Techinal Note 42.

Salisbury, M. 1995. Command and Control Simulation Interface Language (ccsil): Status update. *In: Proceedings of the the 12th distributed interactive simulation workshop.*

Sponsored by STRICOM and the Institute for Simulation and Training (IST) at the University of Central Florida.

Schwamb, K. B., Koss, F. V., & Keirsey, D. 1994 (May). Working with ModSAF: Interfaces for programs and users. *In: Proceedings of the fourth conference on computer generated forces and behavioral representation.*

van Lent, M. 1995. *Planning and learning in a complex domain.* Tech. rept. The University of Michigan, Department of Electrical Engineering and Computer Science.

Wray, R. E. 1995. *A general framework for means-ends analysis.* Tech. rept. The University of Michigan, Department of Electrical Engineering and Computer Science.

10. Author's Biography

Paul E. Nielsen is an assistant research scientist working on the Intelligent Forces Project at the Artificial Intelligence Laboratory of the University of Michigan. He received his Ph.D. from the University of Illinois in 1988. Prior to joining the University of Michigan he worked at the GE Corporate Research and Development Center. His research interests include intelligent agent modeling, qualitative physics, machine learning, and time constrained reasoning.

Recursive Agent and Agent-group Tracking in a Real-time, Dynamic Environment

Milind Tambe
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
tambe@isi.edu
WWW:http://www.isi.edu/soar/tambe

Abstract

Agent tracking is an important capability an intelligent agent requires for interacting with other agents. It involves monitoring the observable actions of other agents as well as inferring their unobserved actions or high-level goals and behaviors. This paper focuses on a key challenge for agent tracking: recursive tracking of individuals or groups of agents. The paper first introduces an approach for tracking recursive agent models. To tame the resultant growth in the tracking effort and aid real-time performance, the paper then presents *model sharing*, an optimization that involves sharing the effort of tracking multiple models. Such shared models are dynamically unshared as needed — in effect, a model is selectively tracked if it is dissimilar enough to require unsharing. The paper also discusses the application of recursive modeling in service of deception, and the impact of sensor imperfections. This investigation is based on our on-going effort to build intelligent pilot agents for a real-world synthetic air-combat environment.[1]

1 Introduction

In dynamic, multi-agent environments, an intelligent agent often needs to interact with other agents to achieve its goals. *Agent tracking* is an important requirement for intelligent interaction. It involves monitoring other agents' observable actions as well as inferring their unobserved actions or high-level goals, plans and behaviors.

Agent tracking is closely related to plan recognition(Kautz & Allen 1986; Azarewicz *et al.* 1986), which involves recognizing agents' plans based on observations of their actions. The key difference is that plan-recognition efforts typically focus on tracking a narrower (plan-based) class of agent behaviors, as seen in static, single-agent domains. Agent tracking, in contrast, can involve tracking a broader mix of goal-driven and reactive behaviors(Tambe & Rosenbloom 1995). This capability is important for dynamic environments where agents do not rigidly follow plans.

This paper focuses on the issues of recursive agent and agent-group tracking. Our investigation is based on an on-going effort to build intelligent pilot agents for simulated air-combat(Tambe *et al.* 1995). These pilot agents execute missions in a simulation environment called ModSAF, that is being commercially developed for the military(Calder *et al.* 1993). ModSAF provides a synthetic yet real-world setting for studying a broad range of challenging issues in agent tracking. By investigating agents that are successful at agent tracking in this environment, we hope to extract some general lessons that could conceivably be applied in other synthetic or robotic multi-agent environments(Kuniyoshi *et al.* 1994; Bates, Loyall, & Reilly 1992).

For an illustrative example of agent tracking in the air-combat simulation environment, consider the scenario in Figure 1. The pilot agent **L** in the light-shaded aircraft is engaged in combat with pilot agents **D** and **E** in the dark-shaded aircraft. Since the aircraft are far apart, **L** can only see its opponents' actions on radar (and vice versa). In Figure 1-a, **L** observes its opponents turning their aircraft in a coordinated fashion to a collision course heading (i.e., with this heading, they will collide with **L** at the point shown by **x**). Since the collision course maneuver is often used to approach one's opponent, **L** infers that its opponents are aware of its (**L**'s) presence, and are trying to get closer to fire their missiles. However, **L** has a missile with a longer range, so **L** reaches its missile range first. **L** then turns its aircraft to point straight at **D**'s aircraft and fires a radar-guided missile at **D** (Figure 1-b). Subsequently, **L** executes a 35° *fpole* turn away from **D**'s aircraft (Figure 1-c), to provide radar guidance to its missile, while slowing its rate of approach to enemy aircraft.

While neither **D** nor **E** can observe this missile on their radar, they do observe **L**'s pointing turn followed

[1]I thank Paul Rosenbloom and Ben Smith for detailed feedback on this effort. Thanks also to Lewis Johnson, Piotr Gmytrasiewicz and the anonymous reviewers for helpful comments. This research was supported under subcontract to the University of Southern California Information Sciences Institute from the University of Michigan, as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Research Laboratory.

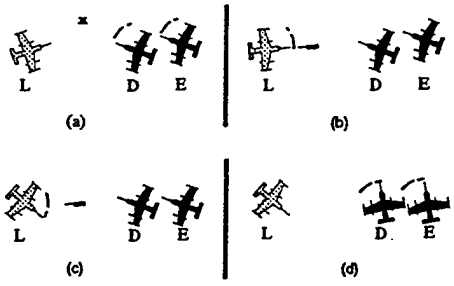


Figure 1: Pilot agents D and E are attacking L. An arc on an aircraft's nose shows its turn direction.

by its fpole turn. They track these to be part of L's missile firing behavior, and infer a missile firing. Therefore, they attempt to evade this missile by executing a 90° *beam* turn (Figure 1-d). This causes their aircraft to become invisible to L's radar. Deprived of radar guidance, L's missile is rendered harmless. Meanwhile, L tracks its opponents' coordinated beam turn in Figure 1-d, and prepares counter-measures in anticipation of the likely loss of its missile and radar contact.

Thus, the pilot agents need to continually engage in agent tracking. They need to track their opponents' actions, such as turns, and infer unobserved actions and high level goals and behaviors, such as the fpole, beam or missile firing behaviors. This paper focuses on two key issues in agent tracking in this environment:

- *Recursive agent tracking:* Pilot agents continually influence each other's behaviors, creating a need for recursive tracking. For instance, in Figure 1-d, to successfully track D's beam, L must also recursively track how D is likely to be tracking L's own actions — that D is aware of L's missile firing, and it is beaming in response. Such recursive tracking may also be used in service of deception, and in addressing other agents' realistic sensor (radar) limitations.
- *Agent group tracking:* An agent may need to track coordinated (or uncoordinated) activities of a group of agents, e.g., as just seen, L needed to track two coordinated opponents.

To address these issues, this paper first presents an approach for recursive tracking of an individual or a groups of agents. This approach builds upon RESC, a technique for real-time tracking of flexible and reactive behaviors of individual agents in dynamic environments. RESC is a real-time, reactive version of the *model tracing* technique used in intelligent tutoring systems — it involves executing a model of the tracked agent, and matching predictions with actual observations(Anderson *et al.* 1990; Ward 1991; Hill & Johnson 1994).

Unfortunately, recursive agent-group tracking leads to a large growth in the number of models. Executing all of these models would be in general highly problematic. The problem is particularly severe for a pilot agent, given that it has to track opponents' maneuvers and counter them in real-time, e.g., by going beam to evade a missile fired at it. Thus, for executing recursive models (and for a practical investigation of recursive tracking), optimizations for real-time performance are critical. Previous work on optimizations for agent tracking has mostly focused on *intra-model* (within a single model) optimizations, e.g., heuristic pruning of irrelevant operators(Ward 1991) restricted backtrack search(Tambe & Rosenbloom 1995), and abstraction(Hill & Johnson 1994). In contrast, this paper proposes *inter-model* (across multiple models) optimizations. It introduces an inter-model optimization called *model sharing*, which involves sharing the effort of tracking multiple models. Shared models are dynamically unshared when required. In essence, a model is selectively tracked if it is dissimilar enough to warrant unsharing. The paper subsequently discusses the application of recursive models in service of deception. This analysis is followed up with some supportive experiments.

The descriptions in this paper assume the perspective of the automated pilot agent L, as it tracks its opponents. They also assume *ideal* sensor conditions, where agents can perfectly sense each others' maneuvers, unless otherwise mentioned. Furthermore, the descriptions are provided in concrete terms using implementations of a pilot agent in a system called TacAir-Soar(Tambe *et al.* 1995), built using the Soar architecture(Newell 1990; Rosenbloom *et al.* 1991). We assume some familiarity with Soar's problem-solving model, which involves applying operators to states to reach a desired state.

2 Recursive Agent Tracking

One key idea in RESC is the uniform treatment of an agent's generation of its own behavior and tracking of other agent's behaviors. As a result, the combination of architectural features that enable an agent to generate flexible goal-driven and reactive behaviors are reused for tracking others' flexible and reactive behaviors. This uniformity is extended in this section in service of recursive agent tracking.

To illustrate this idea, we first describe L's generation of its own behaviors, using the situation in Figure 1-d, just before the agents lose radar contact with each other. Figure 2-a illustrates L's operator hierarchy when executing its fpole. Here, at the top-most level, L is executing its mission — to defend against intruders — via the *execute-mission* operator. Since the termination condition of this operator — completion of L's mission — is not yet achieved, a subgoal is generated.[2]

[2]If an operator's termination conditions remain unsatisfied, a subgoal gets created. If these termination conditions are satisfied by future state changes, then the operator and all its subgoals are terminated.

Different operators are available in this subgoal, such as *follow-flight-path*, *intercept*, and *run-away*. L selects the *intercept* operator to combat its opponent D. In service of *intercept*, L applies the *employ-missile* operator in the next subgoal. Since a missile has been fired, the *fpole* operator is selected in the next subgoal to guide the missile with radar. In the final subgoal *maintain-heading* is applied, causing L to maintain heading (Figure 1-d). All these operators, used for generating L's own actions, will be denoted with the subscript L, e.g., $fpole_{L}$. $Operator_{L}$ will denote an arbitrary operator of L. $State_{L}$ will denote the global state shared by all these operators. Together, $state_{L}$ and the $operator_{L}$ hierarchy constitute L's model of its present dynamic self, referred to as $model_{L}$.

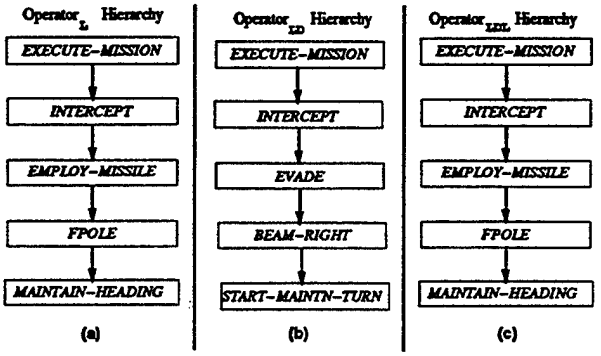


Figure 2: (a) $Model_{L}$; (b) $Model_{LD}$; (c) $Model_{LDL}$.

$Model_{L}$ supports L's flexible/reactive behaviors, given Soar's architectural apparatus for operator selection and termination(Rosenbloom *et al.* 1991). L reuses this apparatus in tracking its opponents' behaviors. Thus, L uses a hierarchy such as the one in Figure 2-b to track D's behaviors. Here, the hierarchy represents L's model of D's current operators in the situation in Figure 1-d. These operators are denoted with the subscript LD. This $operator_{LD}$ hierarchy, and the $state_{LD}$ that goes with it, constitute L's model of D or $model_{LD}$. $Model_{LD}$ obviously cannot and does not directly influence D's actual behavior, it only tracks D's behavior. For instance, in the final subgoal, L applies the *start-&-maintain-turn*$_{LD}$ operator, which does not cause D to turn. Instead, this operator predicts D's action and matches the prediction with D's actual action. Thus, if D starts turning right towards beam, then there is a match with $model_{LD}$ — L believes that D is turning right to beam and evade its missile, as indicated by other higher-level operators in the $operator_{LD}$ hierarchy. Note that, in reality, from L's perspective, there is some ambiguity in D's right turn in Figure 1-d — it could be part of a big 150° turn to run away given L's missile firing. To resolve such ambiguity, L adopts several techniques, such as assuming the worst-case hypothesis about its enemy, which in this case is that D is beaming rather than running away. We will not discuss RESC's ambiguity resolution any further in this paper (see (Tambe & Rosenbloom 1995) for more details).

Thus, with the RESC approach, L tracks D's behaviors by continuously executing the $operator_{LD}$ hierarchy, and matching it against D's actions. To recursively track its own actions from D's perspective, L may apply the same technique to its recursive $model_{LDL}$ (L's model of D's model of L) as shown in Figure 2-c. $Model_{LDL}$ consists of an $operator_{LDL}$ hierarchy and $state_{LDL}$. The important point here is the uniform treatment of the $operator_{LDL}$ hierarchy — on par with $operator_{LD}$ and $operator_{L}$ hierarchies — to support the tracking of flexible and reactive behaviors. L tracks $model_{LDL}$ by matching predictions with its own actions. Further recursive nesting leads to the tracking of $model_{LDLD}$ and so on. To track additional opponents, e.g., the second opponent E, L tracks additional models, such as $model_{LE}$. L may also track $model_{LEL}$, $model_{LED}$, $model_{LDE}$ etc for recursive tracking.

Recursive tracking is key to tracking other agents' behaviors in interactive situations. Thus, it is L's recursive tracking of $fpole_{LDL}$ which indicates a missile firing to $model_{LD}$, and causes the selection of *evade-missile*$_{LD}$ to track D's missile evasion. Note that in Figure 2-c, ambiguity resolution in $model_{LDL}$ leads to an $operator_{LDL}$ hierarchy that is identical to the $operator_{L}$ hierarchy. One key ambiguity resolution strategy is again the worst-case assumption — given ideal sensor situations, L assumes D can accurately track L's behaviors. Thus, among possible options in the $operator_{LDL}$ hierarchy, the one identical to $operator_{L}$ gets selected. However, these hierarchies may not always be identical and the differences between them may be exploited in service of deception at least in adversarial situations. These possibilities are discussed in more detail in Section 5.1.

3 Executing Models in Real-time

Unfortunately, the recursive tracking scheme introduced in the previous section points to an exponential growth in the number of models to be executed. In general, for N opponents, and r levels of nesting (measured with $r = 1$ for $model_{L}$, $r = 2$ for $model_{LD}$, and so on), the pilot agent L may need to execute: $\sum_{i=0}^{r-1} N^i$ models (which is r for $N = 1$, but $\frac{N^r-1}{N-1}$ for $N > 1$). This is clearly problematic given the likely scale-up in N. In particular, given its limited computational resources, L may be unable to execute relevant operators from all its models in real-time, jeopardizing its survival. In fact, as seen in Section 6, L may run into resource contention problems while executing just five models — indicating possible difficulties even for small N and r.

Thus, optimizations involving some form of selective tracking appear necessary for real-time execution of these models. Yet, such selectivity should not cause

an agent to be completely ignorant of critical information that a model may provide (e.g., an agent should not be ignorant of an opponent's missile firing). To this end, this paper focuses on an optimization called *model sharing*. The overall motivation is that if there is a model$_{\mathbf{y}}$ that is near-identical to a model$_{\mathbf{x}}$, then model$_{\mathbf{y}}$'s states and operators can be shared with those of model$_{\mathbf{x}}$. Thus, model$_{\mathbf{y}}$ is tracked via the execution of model$_{\mathbf{x}}$, reducing the tracking effort in half. Model$_{\mathbf{y}}$ may be dynamically unshared from model$_{\mathbf{x}}$ if it grows significantly dissimilar. Thus, a model is selectively executed based on its dissimilarity with other models.

For an illustration of this optimization, consider model$_{\mathbf{L}}$ and model$_{\mathbf{LDL}}$, as shown in Figure 2. The operator$_{\mathbf{LDL}}$ hierarchy can be shared with the operator$_{\mathbf{L}}$ hierarchy since the two are identical. (In low-level implementation terms, sharing an operator$_{\mathbf{L}}$ involves adding a pointer indicating it is also a part of model$_{\mathbf{LDL}}$). Furthermore, information in state$_{\mathbf{LDL}}$ is shared with state$_{\mathbf{L}}$. Thus, L essentially executes operators from only one model, instead of two.

Given the efficiency benefits from sharing, it is often useful to abstract away from some of the differences between models in order to enable sharing. However, such abstraction may not be possible for some static and/or dynamic aspects of the models. One important aspect relates to private information. In particular, in their unshared incarnations, models have their indices organized so as to prevent a breach of privacy, e.g., model$_{\mathbf{LD}}$ can access information in model$_{\mathbf{LDL}}$, but not model$_{\mathbf{L}}$. Model sharing could potentially breach such privacy. Thus, for instance, if state$_{\mathbf{L}}$ maintains secret missile information, sharing it with state$_{\mathbf{LDL}}$ would allow model$_{\mathbf{LD}}$ to access that secret. To prevent model sharing from breaching such privacy, some aspects of the shared models may be explicitly maintained in an unshared fashion. Thus, if L's missile range is (a secret) 30 miles, but L believes D believes it is 50 miles, then the missile range is maintained separately on state$_{\mathbf{L}}$ and state$_{\mathbf{LDL}}$. Figure 3 shows the resulting shared models with an unshared missile range.

Such sharing among models is related to the sharing of belief spaces in the SNePS belief representation system(Shapiro & Rapaport 1991). One key difference is *dynamic model unsharing*. In particular, while some of aspects of the models are static (e.g., the statically unshared missile ranges above), other aspects, particularly those relating to operators, are highly dynamic. As a result, shared components may need to be dynamically unshared when dissimilar. Ideally, any two models could be merged (shared) when they are near-identical, and dynamically unshared in case of differences. This would be ideal selectivity — a model is tracked if it requires unsharing. However, in practice, both unsharing and merging may involve overheads. Thus, if an agent greedily attempts to share any two models whenever they appear near-identical, it could face very heavy overheads. Instead, it has to selectively share two models over a time period Δ so that the savings from sharing outweigh the cost of dynamic unsharing and re-merging during Δ. In particular, suppose there are two models, model$_{\mathbf{x}}$ and model$_{\mathbf{y}}$ that are unshared over n sub-intervals $\delta_1, \delta_2, \ldots, \delta_n$ of Δ, but shared during the rest of Δ. Further, suppose $cost_\theta(\mathcal{M})$ is the cost of executing a model $\mathcal{M}$ over a time interval θ; cost(unshare) and cost(merge) are the overheads of unsharing and merging respectively; and $cost_\theta$(detect) is the cost incurred during θ of deciding if shared models need to be unshared or if unshared models can be merged (this may potentially involve comparing two different models and deciding if sharing is cost-effective). Then, in sharing model$_{\mathbf{y}}$ with model$_{\mathbf{x}}$ during Δ, benefits outweight costs iff:

$$cost_\Delta(model_{\mathbf{y}}) - \sum_{i=\delta_1}^{\delta_n} cost_i(model_{\mathbf{y}}) > n \times cost(unshare) + n \times cost(merge) + cost_\Delta(detect) \quad \ldots\ldots\ldots (1)$$

While, ideally, agents may themselves evaluate this equation, our agents are unable to do so at present. Therefore, candidate categories of models — with high likelihood of sharing benefits outweighing costs — are supplied by hand. Nonetheless, agents do determine specific models within these categories that may be shared, and implement the actual sharing and dynamic unsharing. The categories are:

1. Models of distinct agents at the recursive depth of $r = 2$: If a group of agents, say D and E, together attack L, model$_{\mathbf{LD}}$ and model$_{\mathbf{LE}}$ may be possibly shared. Thus, if all models of its N opponents are shared, L may need track only one model at $r = 2$; if not shared, L may track N models at $r = 2$.
2. Recursive models of a single agent at $r \geq 3$: For instance, model$_{\mathbf{LDL}}$ and model$_{\mathbf{LEL}}$ may be shared with model$_{\mathbf{L}}$. Similarly, model$_{\mathbf{LDE}}$ may be shared with model$_{\mathbf{LE}}$ or model$_{\mathbf{LELE}}$, etc. Models at recursive depth $r \geq 3$ may all be shared with models at $r = 1$ or 2. If all such models are shared, L may need to track no models at $r \geq 3$.

The end result is that an agent L may track a group

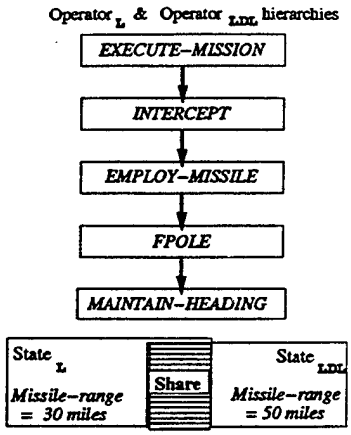


Figure 3: Sharing model$_{\mathbf{L}}$ with model$_{\mathbf{LDL}}$.

of N agents (with sharable models) with just two models — model_{L} at $r = 1$, and one model at $r = 2$, and the rest are all shared — instead of $O(N^r)$ models. If the models of N agents are not sharable, then it may still need just $N + 1$ models, given the sharing in the recursion hierarchy. Thus, sharing could provide substantial benefits in tracking even for small N and r. In the following, Section 4 examines in more detail the model sharing within a group, and Section 5 examines sharing within and across recursion hierarchies.

4 Sharing in Agent-Group Tracking

Agents that are part of a single group often act in a coordinated fashion — executing similar behaviors and actions — and thus provide a possible opportunity for model sharing. For instance, if D and E are attacking L in a coordinated fashion, they may fly in formation, execute similar maneuvers etc. However, their actions are not perfectly identical — there are always some small delays in coordination for instance — which can be a possible hinderence in sharing. If the delays and differences among the agents' actions are small, they need to be abstracted away, to facilitate model sharing. Yet, such abstraction should allow tracking of essential group activities.

To this end, one key idea to track an agent-group is to track only a single *paradigmatic agent* within the group. Models of all other agents within the group are then shared with the model of this paradigmatic agent. Thus, a whole group is tracked by tracking a single paradigmatic agent. For example, suppose L determines one agent in the attacking group, say D, to be the paradigmatic agent. It may then only track model_{LD}, and share other models, such as model_{LE} with model_{LD}, reducing its the tracking burden.

Such model sharing needs to selective, if benefits are to outweigh costs. In this case, the following domain-specific heuristics help tilt the balance in favor of sharing by reducing the cost of detection, merging and unsharing:

- Cost(detect): This involves detecting two or more agents (opponents) to be part of a group with sharable models. Such a group is detected at low cost by testing the agents' physical proximity and direction of movement. If these are within the ranges provided by domain experts, the corresponding agents' models are shared. If the agents move away from each other (outside of this range) their models are unshared. Once outside this range, no attempt is made at model sharing — such agents are likely to be engaged in dissimilar activities, and even if their models are found to be near-identical, they are likely to be so for a short time period.
- Cost(merge): Merging involves the cost of selecting a paradigmatic agent within the group. It may be possible to select an agent at random from the group for this role. However, an agent in some prominent position, such as in front of the group, is possibly a better fit for the role of a paradigmatic agent, and can also be picked out at a low cost. In air-combat simulation, an agent in such a position is typically the leader of the group of attacking opponents. It initiates maneuvers, and others follow with a small time-lag. The group leader is thus ideal as a paradigmatic agent. Note that a dynamic change in the paradigmatic agent does not cause unsharing.
- Cost(Unshare): Unsharing, however, has a rather high cost. For instance, once D and E are detected to have unshared models, a completely new model_{LE} is constructed. Here, the entire state_{LD} has to be copied to state_{LE}.

The end result is that a particular agent's model is selectively executed when the agent breaks away from the coordinated group. Otherwise, its model is merged with the paradigmatic agent's model.

5 Sharing in Recursion Hierarchy

Models of a single agent across a recursion hierarchy are likely to be near-identical to each other, and thus they form the second category of models that may allow sharing. We have so far limited our investigation of sharing/unsharing to models with $r \leq 3$, and specifically to different models of L, such as $\text{model}_{\text{LDL}}$ and $\text{model}_{\text{LEL}}$ at $r = 3$, with model_{L} at $r = 1$. Other models, including those at deeper levels of nesting ($r \geq 4$) are never unshared. For instance, $\text{model}_{\text{LDLD}}$ is never unshared from model_{LD}. The motivation for this restriction is in part that in our interviews with domain experts, references to unshared models at $r \geq 4$ have rarely come up. In part, this also reflects the complexity of such unsharing, and it is thus an important issue to be addressed in future work.

To understand the cost-benefit tradeoffs of sharing recursive models, it is first useful to understand how sharing and unsharing may actually occur. One general technique for accomplishing sharing in the recursion hierarchy is to first let model_{L} generate its operator hierarchy. As the hierarchy is generated, if $\text{model}_{\text{LDL}}$ agrees with an $\text{operator}_{\text{L}}$ — that is, it would have generated an identical $\text{operator}_{\text{LDL}}$ given $\text{state}_{\text{LDL}}$ — then it ($\text{model}_{\text{LDL}}$) "votes" in agreement. This "vote" indicates that that particular $\text{operator}_{\text{L}}$ from model_{L} is now shared with $\text{model}_{\text{LDL}}$. This essentially corresponds to the worst case strategy introduced in section 2 — given a choice among $\text{operators}_{\text{LDL}}$, the one that is identical to $\text{operator}_{\text{L}}$ is selected and shared.

Thus, the detection/merging cost is low, since this can be accomplished without an extensive comparison of models. Furthermore, the savings from model sharing are substantial — as discussed below, unsharing occurs over small time periods. Furthermore, the unsharing cost is low, since it does not involve state copying. Thus, sharing benefits appear to easily outweigh

its costs.

Unsharing actually occurs because of differences between state_{L} and $\text{state}_{\text{LDL}}$. Due to these differences, the recursive $\text{model}_{\text{LDL}}$ cannot generate an operator that is shared with $\text{operator}_{\text{L}}$. There is then unsharing of the operator hierarchies in $\text{model}_{\text{LDL}}$ and model_{L}, which may be harnessed in service of deceptive (or other) tactics. In the following, Subsection 5.1 focuses on one general strategy for such deception. Subsection 5.2 focuses on a special class of differences between recursive states — caused by sensor imperfections — and the deceptive maneuvers possible due to those differences.

5.1 Deception

Due to differences between $\text{state}_{\text{LDL}}$ and state_{L}, $\text{model}_{\text{LDL}}$ may generate an $\text{operator}_{\text{LDL}}$ that cannot be shared in the $\text{operator}_{\text{L}}$ hierarchy. This indicates to L that D expects L to be engnaged in a different maneuver ($\text{operator}_{\text{LDL}}$) than the one it is actually executing ($\text{operator}_{\text{L}}$). In such cases, L may attempt to deceive D by abandoning its on-going maneuver and "playing along" with what it believes to be D's expectations.

To understand this deceptive strategy, consider the following case of L's deceptive missile firing. Let us go back to the situation in Figure 1-a, although now, assume that state_{L} maintains a secret missile range of 30 miles, while $\text{state}_{\text{LDL}}$ maintains the range to be 50 miles. The missile range is noted in the unshared portions of the states as shown in Figure 3. At a range of 50 miles — given that $\text{state}_{\text{LDL}}$ notes the missile range to be 50 miles — $\text{model}_{\text{LDL}}$ suggests the execution of a *employ-missile*$_{\text{LDL}}$ operator. This causes unsharing with operators in model_{L}. *Employ-missile*$_{\text{LDL}}$ subgoals into *get-steering-circle*$_{\text{LDL}}$, indicating a turn to point at target, as shown in Figure 1-b.

These operators suggest actions for L in order to deceive its opponent. L may execute deceptive $\text{operators}_{\text{L}}$ that create the external actions suggested by $\text{operator}_{\text{LDL}}$ *without actually launching a missile.* L therefore executes a *employ-missile-deceptive*$_{\text{L}}$ operator. This subgoals into the *get-steering-circle-deceptive*$_{\text{L}}$ operator. This causes the next subgoal, of *start-&-maintain-turn*$_{\text{L}}$ in model_{L} which actually causes L to turn to point at its target, D. This difference in model_{L} and $\text{model}_{\text{LDL}}$ causes some unsharing in their operator hierarchies, as shown in Figure 4. After pointing at target, $\text{model}_{\text{LDL}}$ executes the *fpole*$_{\text{LDL}}$ operator — that is, L believes that D is expecting L's fpole to support an actual missile in the air. L executes *fpole-deceptive*$_{\text{L}}$ without actually firing a missile. Thus, with a deceptive maneuver, L convinces D that it has fired a missile at a much longer range, without actually firing one — forcing D to go on the defensive by turning towards beam.

L can employ a whole class of such deceptive maneuvers by going along with $\text{model}_{\text{LDL}}$'s expectation, as it did here. This is essentially a general strategy for deceptive maneuvers, which is instantiated with particular deceptive maneuvers in real-time. Yet, this is only a first step towards a full-fledged deceptive agent. There are many other deceptive techniques and issues that remain unresolved, e.g., determining whether engaging in deception would lead to a globally sub-optimal behavior.

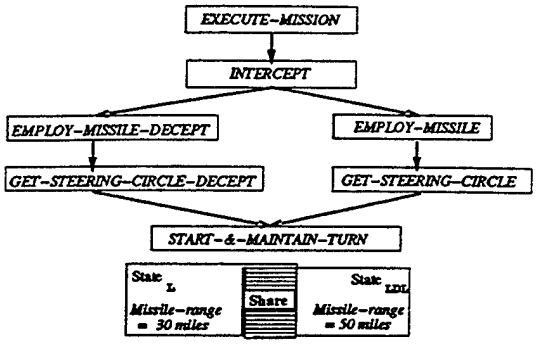


Figure 4: Deceptive missile firing: $\text{operator}_{\text{L}}$ and $\text{operator}_{\text{LDL}}$ hierarchies are dynamically unshared.

5.2 Sensor Imperfections

Realistic radar imperfections in this domain also lead to unsharing among recursive models. It is useful to examine these in some detail, since these are illustrative of the types of differences that are expected to arise in other domains where agents have realistic sensors. To this end, it is useful to classify the different situations resulting from these imperfections as shown in Figure 5. As a simplification, these situations describe L's perspective as it interacts with a single opponent for D, and limited to $r \leq 3$. Figure 5-a focuses on an agent's awareness of another's presence. In the figure, Aware<BAZ> denotes someone's awareness of an agent named BAZ. Furthermore, subscript L indicates L's own situation, a subscript LD indicates L's tracking of D, a subscript LDL indicates L's recursive tracking of D's tracking of L. Thus, the first branch point in 5-a indicates whether L is aware of D's presence ($+\text{Aware}_{\text{L}}$<D>) or unaware ($-\text{Aware}_{\text{L}}$<D>). If $-\text{Aware}_{\text{L}}$<D> then L can not track D's awareness. If $+\text{Aware}_{\text{L}}$<D>, then L may believe that D is aware of L's presence ($+\text{Aware}_{\text{LD}}$<L>) or unaware ($-\text{Aware}_{\text{LD}}$<L>). If $+\text{Aware}_{\text{LD}}$<L>, then L may have beliefs about D's beliefs about L's awareness: $+\text{Aware}_{\text{LDL}}$<D> or $-\text{Aware}_{\text{LDL}}$<D>.

While an agent may be aware of another, it may not have accurate sensor information of the other agent's actions, specifically, turns, climbs and dives. For instance, in Figure 1-d, $+\text{Aware}_{\text{L}}$<D>, yet L loses radar contact due to D's beam. Figure 5-b classifies these situations. Here, $+\text{Sense}_{\text{L}}$<D> refers

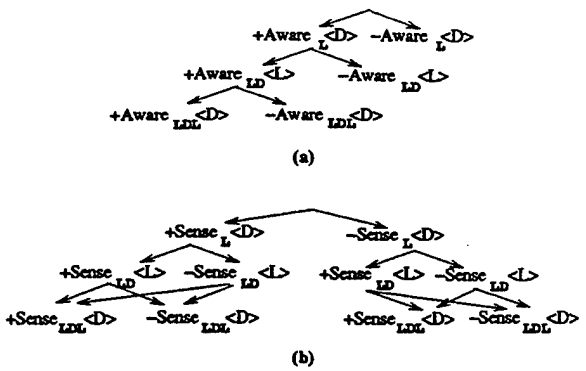


Figure 5: Classifications by: (a) awareness; (b) accuracy of sensor information.

to situations where L believes it has accurate sensor information of D's actions, while $-\text{Sense}_{\mathbf{L}}$<D> refers to situations as in Figure 1-d, where it does not. In either case, L may believe that D either has accurate sensor information of L's actions ($+\text{Sense}_{\mathbf{LD}}$<L>) or not ($-\text{Sense}_{\mathbf{LD}}$<L>). Thus, in Figure 1-d, while $-\text{Sense}_{\mathbf{L}}$<D>, L also believes D has lost radar contact due to its 90° beam turn ($-\text{Sense}_{\mathbf{LD}}$<L>). Recursion continues with $+\text{Sense}_{\mathbf{LDL}}$<D> and $-\text{Sense}_{\mathbf{LDL}}$<D>.

Based on the above classification, L's perspective of a situation may be described as a six-tuple. For instance, Figures 1-a to 1-c may be described as ($+\text{Aware}_{\mathbf{L}}$<D>, $+\text{Aware}_{\mathbf{LD}}$<L>, $+\text{Aware}_{\mathbf{LDL}}$<D>, $+\text{Sense}_{\mathbf{L}}$<D>, $+\text{Sense}_{\mathbf{LD}}$<L>, $+\text{Sense}_{\mathbf{LDL}}$<D>). This is the previously introduced *ideal* sensor situation, with "+" awareness and sensor accuracy. Based on the six-tuple, 64 such situations seem possible. However, many are ruled out — if an agent is unaware of another, it cannot have accurate sensor information regarding that agent — reducing the number of possible situations to 15.

Within these 15, we have so far examined unsharing and deception in the context of one situation, namely the ideal situation. We now briefly examine the unsharing and deception possible in the remaining 14 situations. Among these 14, there are three that typically arise in the initial portions of the combat where L believes D is unaware of L ($-\text{Aware}_{\mathbf{LD}}$<L>). For instance, L may have seen D by virtue of its longer range radar, but it may have assumed that D is still unaware due to its shorter range radar: ($+\text{Aware}_{\mathbf{L}}$<D>, $-\text{Aware}_{\mathbf{LD}}$<L>, $-\text{Aware}_{\mathbf{LDL}}$<D>, $+\text{Sense}_{\mathbf{L}}$<D>, $-\text{Sense}_{\mathbf{LD}}$<L>, $-\text{Sense}_{\mathbf{LDL}}$<D>). In all these cases, $\text{model}_{\mathbf{LDL}}$ is null, and thus the question of sharing with $\text{model}_{\mathbf{L}}$ does not arise. Suppose as the aircraft move even closer, D engages in the collision course maneuver, which allows L to conclude that $+\text{Aware}_{\mathbf{LD}}$<L>. Here, there are two possibilities. First, if $-\text{Aware}_{\mathbf{LDL}}$<D>, i.e., L believes D believes L is unaware of D, there is much greater dissimilarity between $\text{model}_{\mathbf{LDL}}$ and $\text{model}_{\mathbf{L}}$. $\text{Model}_{\mathbf{LDL}}$ now predicts that L will not engage in combat with D, i.e., there will be unsharing even with the $\textit{intercept}_{\mathbf{L}}$ operator. Once again, L may deceive D by acting consistent with $\text{model}_{\mathbf{LDL}}$'s expectation, and not turn towards D. This is similar to the deceptive strategy introduced in Section 5.1. L may then wait till D gets closer and then turn to attack.

The second possibility is $+\text{Aware}_{\mathbf{LDL}}$<D>. In this case, we return to the *ideal* situation in Figures 1-a to 1-c, where unsharing is still possible as in Figure 4. Furthermore, even with $+\text{Aware}_{\mathbf{LDL}}$<D>, there are situations with $-\text{Sense}_{\mathbf{LD}}$<L>, where L believes D cannot sense L's actions. In such cases, L may engage in deception by deliberately *not* acting consistent with $\text{model}_{\mathbf{LDL}}$'s expectations, e.g., diving when $\text{model}_{\mathbf{LDL}}$ does not expect such a dive. Such deliberate unsharing is another type of deceptive strategy that among many others, is one we have not examined in detail so far.

6 Experimental Results

To understand the effectiveness of the agent tracking method introduced here, we have implemented an experimental variant of TacAir-Soar(Tambe *et al.* 1995). The original TacAir-Soar system contains about 2000 rules, and automated pilots based on it have participated in combat exercises with expert human pilots(Tambe *et al.* 1995). Our experimental version — created since it employs an experimental agent tracking technology — contains about 950 of these rules. This version can recursively track actions of individuals or groups of opponents while using the model sharing optimizations, and engaging in deception. Proven techniques from this experimental version are transferred back into the original TacAir-Soar.

Table 1 presents experimental results for typical simulated air-combat scenarios provided by the domain experts. Column 1 indicates the number of opponents (N) faced by our TacAir-Soar-based agent L. Column 2 indicates whether the opponents are engaged in a coordinated attack. Column 3 shows the actual maximum number of models used in the combat scenarios with the optimizations (excluding temporary model unsharing in service of deception). The numbers in parentheses are projected number of models — $2N+1$ — without the model sharing optimization (the actual number without sharing should be $O(N^r)$, but we exclude the permanently shared models from this count — see Section 5). With optimizations, as expected, the number of models is $N+1$ when opponents are not coordinated, and just two when the opponents are coordinated. Column 4 shows the actual and projected number of operator executions. The projected number is calculated assuming $2N+1$ models. Column 5 shows a two to four fold reduction (projected/actual) in the number of operators. Savings are higher with coordinated opponents. L is usually successful in real-time

tracking in that it is able to track opponents' behaviors rapidly enough to be able to respond to them. L is unsuccessful in real-time tracking in the case of four uncoordinated opponents (with 5 models), and it gets shot down (hence fewer total operators than the case of 2 opponents). This failure indicates that our optimizations *have helped* — without them, L could have failed in all cases of 2 or 4 opponents since they involve 5 or more projected models. It also indicates that L may need additional optimizations.

N	Coord?	Actual(projct) num max model	Actual(projct) total operatrs	Reduction (proj/act)
1	–	2 (3)	143(213)	1.5
2	No	3 (5)	176(314)	1.8
4	No	5 (9)	148(260)	1.8
2	Yes	2 (5)	109(251)	2.3
4	Yes	2(9)	105(407)	3.9

Table 1: Improvements due to model sharing.

7 Summary

This paper focused on real-time recursive tracking of agents and agent-groups in dynamic, multi-agent environments. Our investigation was based on intelligent pilot agents in a real-world synthetic air-combat environment, already used in a large-scale operational military exercise(Tambe *et al.* 1995). Possible take-away lessons from this investigation include:

- Address recursive agent tracking via a uniform treatment of the generation of flexible/reactive behaviors, as well as of tracking and recursive tracking.
- Alleviate tracking costs via model sharing — with selective unsharing in situations where models grow sufficiently dissimilar.
- Track group activities by tracking a paradigmatic agent.
- Exploit differences in an agent's self model and its recursive self model in service of deception and other actions.

One key issue for future work is understanding the broader applicability of these lessons. To this end, we plan to explore the relationships of our approach with formal methods for recursive agent modeling(Gmytrasiewicz, Durfee, & Wehe 1991; Wilks & Ballim 1987). This may help generalize the tracking approach introduced in this paper to other multi-agent environments, including ones for entertainment or education.

References

Anderson, J. R.; Boyle, C. F.; Corbett, A. T.; and Lewis, M. W. 1990. Cognitive modeling and intelligent tutoring. *Artificial Intelligence* 42:7–49.

Azarewicz, J.; Fala, G.; Fink, R.; and Heithecker, C. 1986. Plan recognition for airborne tactical decision making. In *Proceedings of the National Conference on Artificial Intelligence*, 805–811. Menlo Park, Calif.: AAAI press.

Bates, J.; Loyall, A. B.; and Reilly, W. S. 1992. Integrating reactivity, goals and emotions in a broad agent. Technical Report CMU-CS-92-142, School of Computer Science, Carnegie Mellon University.

Calder, R. B.; Smith, J. E.; Courtemanche, A. J.; Mar, J. M. F.; and Ceranowicz, A. Z. 1993. Modsaf behavior simulation and control. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation.*

Gmytrasiewicz, P. J.; Durfee, E. H.; and Wehe, D. K. 1991. A decision theoretic approach to co-ordinating multi-agent interactions. In *Proceedings of International Joint Conference on Artificial Intelligence.*

Hill, R., and Johnson, W. L. 1994. Situated plan attribution for intelligent tutoring. In *Proceedings of the National Conference on Artificial Intelligence.* Menlo Park, Calif.: AAAI press.

Kautz, A., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of the National Conference on Artificial Intelligence*, 32–37. Menlo Park, Calif.: AAAI press.

Kuniyoshi, Y.; Rougeaux, S.; Ishii, M.; Kita, N.; Sakane, S.; and Kakikura, M. 1994. Cooperation by observation – the framework and the basic task pattern. In *Proceedings of the IEEE International Conference on Robotics and Automation.*

Newell, A. 1990. *Unified Theories of Cognition.* Cambridge, Mass.: Harvard Univ. Press.

Rosenbloom, P. S.; Laird, J. E.; Newell, A.; ; and McCarl, R. 1991. A preliminary analysis of the soar architecture as a basis for general intelligence. *Artificial Intelligence* 47(1-3):289–325.

Shapiro, S. C., and Rapaport, W. J. 1991. *Models and minds: knowledge representation for natural language competence.* Cambridge, MA: MIT Press.

Tambe, M., and Rosenbloom, P. S. 1995. Resc: An approach to agent tracking in a real-time, dynamic environment. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).*

Tambe, M.; Johnson, W. L.; Jones, R.; Koss, F.; Laird, J. E.; Rosenbloom, P. S.; and Schwamb, K. 1995. Intelligent agents for interactive simulation environments. *AI Magazine* 16(1).

Ward, B. 1991. *ET-Soar: Toward an ITS for Theory-Based Representations.* Ph.D. Dissertation, School of Computer Science, Carnegie Mellon Univ.

Wilks, Y., and Ballim, A. 1987. Multiple agents and hueristic ascription of belief. In *Proceedings of International Joint Conference on Artificial Intelligence.*

RESC: An Approach for Real-time, Dynamic Agent Tracking

Milind Tambe and Paul S. Rosenbloom
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
Email: {tambe, rosenbloom}@isi.edu
WWW: http://www.isi.edu/soar/{tambe,rosenbloom}

Abstract

Agent tracking involves monitoring the observable actions of other agents as well as inferring their unobserved actions, plans, goals and behaviors. In a dynamic, real-time environment, an intelligent agent faces the challenge of tracking other agents' flexible mix of goal-driven and reactive behaviors, and doing so in real-time, despite ambiguities. This paper presents RESC (REal-time Situated Commitments), an approach that enables an intelligent agent to meet this challenge. RESC's situatedness derives from its constant uninterrupted attention to the *current* world situation — it always tracks other agents' on-going actions in the context of this situation. Despite ambiguities, RESC quickly commits to a single interpretation of the on-going actions (without an extensive examination of the alternatives), and uses that in service of interpretation of future actions. However, should its commitments lead to inconsistencies in tracking, it uses *single-state backtracking* to undo some of the commitments and repair the inconsistencies. Together, RESC's situatedness, immediate commitment, and single-state backtracking conspire in providing RESC its real-time character.

RESC is implemented in the context of intelligent pilot agents participating in a real-world synthetic air-combat environment. Experimental results illustrating RESC's effectiveness are presented.[1]

1 Introduction

In a multi-agent environment, an automated agent often needs to interact intelligently with other agents to achieve its goals. *Agent tracking* — monitoring other agents' observable actions and inferring their unobserved actions, plans, goals and behaviors — is a key capability required to support such interaction.

This paper focuses on agent tracking in real-time, dynamic environments. Our approach is to first build agents that are (reasonably) successful in agent tracking in such environments, and then attempt to understand the underlying principles. Thus, the approach is one of first building an "interesting" system for a complex environment, and then understanding why it does or does not work (see [Hanks *et al.*, 1993] for a related discussion). In step with this approach, we are investigating agent tracking in the context of our on-going effort to build intelligent pilot agents for a real-world synthetic air-combat environment[Tambe *et al.*, 1995]. This environment is based on a commercially developed simulator called ModSAF[Calder *et al.*, 1993], which has already been used in an operational military exercise involving expert human pilots. For an illustrative example of agent tracking in this environment, consider the scenario in Figure 1. It involves two combating pilot agents — **L** in the light-shaded aircraft and **D** in the dark-shaded one.

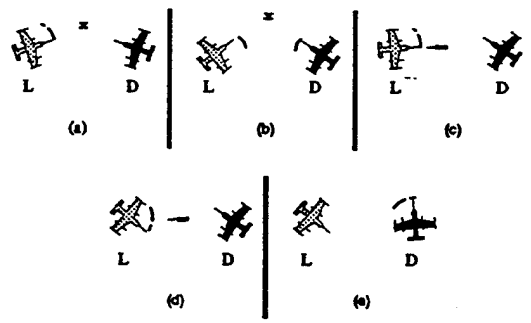


Figure 1: A simulated air-combat scenario. An arc on an aircraft's nose shows its turn direction.

Initially, **L** and **D**'s aircraft are 50 miles apart, so they can only see each other's actions on radar. For effective performance, they have to continually track these actions. Indeed, **D** is able to survive a missile attack by **L** in this scenario due to such tracking, despite the missile being invisible to **D**'s radar. In particular, in Figure 1-a, **D** observes **L** turning its aircraft to a collision-course heading (i.e., at this heading, **L** will collide with **D** at

[1]We thank Rick Lewis and Yasuo Kuniyoshi for helpful feedback. This research was supported under subcontract to the University of Southern California Information Sciences Institute from the University of Michigan, as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Research Laboratory (NRL).

the point shown by x). Since this heading is often used to reach one's missile firing range, D infers the possibility that L is trying to reach this range to fire a missile. In Figure 1-b, D turns its aircraft 15° right. L reacts by turning 15° left, to maintain collision course. In Figure 1-c, L reaches its missile range, points its aircraft at D's aircraft and fires a radar-guided missile. While D cannot see the missile on its radar, it observes L's turn, and infers it to be part of L's missile firing behavior. Subsequently, D observes L executing a 35° turn away from its aircraft (Figure 1-d). D infers this to be an *fpole* turn, typically executed after firing a missile to provide radar guidance to the missile, while slowing the closure between the two aircraft. While D still cannot observe the missile, it is now sufficiently convinced to attempt to evade the missile by turning 90° relative to the direction of L's aircraft (Figure 1-e). This *beam* turn causes D's aircraft to become invisible to L's (doppler) radar. Deprived of radar guidance, L's missile is rendered harmless.

Meanwhile, L tracks D's beam turn in Figure 1-e, and prepares counter-measures in anticipation of the likely loss of both its missile and radar contact.

Thus, the pilot agents need to continually track their opponents' actions, such as turns, and infer unobserved actions, high-level goals and behaviors, such as the fpole, beam or missile firing behaviors. This agent tracking capability is related to plan-recognition[Kautz and Allen, 1986; Azarewicz *et al.*, 1986]. The key difference is that plan-recognition efforts typically focus on tracking a narrower (plan-based) class of agent behaviors, as seen in static, single-agent domains. In particular, they assume that agents rigidly follow plans step-by-step. In contrast, agent tracking involves the novel challenge of tracking a broader mix of goal-driven and reactive behaviors. This capability is important for dynamic environments such as air-combat simulation where agents do not rigidly follow plans — as just seen, pilot agents continually react to each other's maneuvers.

Agent tracking and plan recognition are both part of a larger family of comprehension capabilities that enable an agent to parse a continuous stream of input from its environment, whether it be in the form of natural language or speech or music or simulated radar input, as is the case here (e.g., see [Rich and Knight, 1990, chapter 14]). Resolving ambiguities in the input stream is a key problem when parsing all of these different types of input. One example of the ambiguity faced in agent tracking can be seen in L's turn in Figure 1-c. From D's perspective, L could be turning to fire a missile. Alternatively, L could be beginning a 180° turn to run away from combat. Or L could simply be following its flight plan, particularly if it has a much shorter radar range, and thus is likely unaware of D. Despite such ambiguities, D has to track L's actions with sufficient accuracy so as to respond appropriately. The novel challenge in this domain — at least with respect to previous work in plan recognition — is that the ambiguity resolution has to occur in real-time. As the world rapidly moves on, an agent cannot lag behind in tracking. Thus, if D is late or inaccurate in its tracking of L's missile firing maneuvers in Figure 1-c, it may not evade the missile in time.

This paper describes an approach called RESC (REaltime Situated Commitments) for agent tracking that addresses the above challenges. RESC's situatedness rests on its constant attention to the current world situation, and its tracking of other agents' actions in the context of this situation. Despite its situatedness, RESC does make some commitments about the other agent's unobservable actions, behaviors and goals, and attempts to use those in tracking the agent's future actions. In ambiguous situations, these commitments could be inappropriate and could lead to failures in tracking — in such cases, RESC modifies them on-line, without re-examining past world states. Together, RESC's situatedness, immediate commitments (despite the ambiguities), and its on-line modification of commitments provide RESC its real-time character.

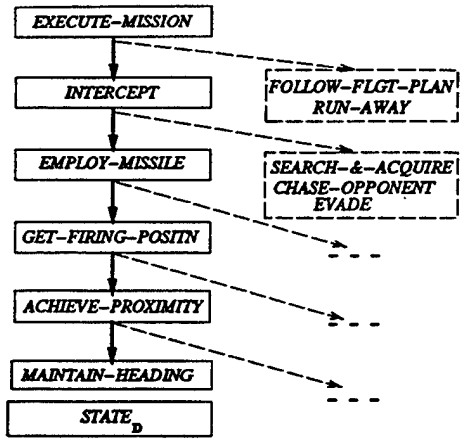
In the following, we first describe the process that RESC employs for tracking other agent's flexible and reactive behaviors (Section 2). This process enables RESC to be situated in its present as it tracks an agent's actions. Subsequently, RESC's ambiguity resolution and real-time properties are described in Section 3. These descriptions are provided in concrete terms, using an implementation of the pilot agents in a system called TacAir-Soar[Tambe *et al.*, 1995], built using the Soar architecture[Newell, 1990; Rosenbloom *et al.*, 1991]. We assume some familiarity with Soar's problem-solving model, which involves applying operators to states to reach a desired state.

2 Tracking Flexible Goal-driven and Reactive Behaviors

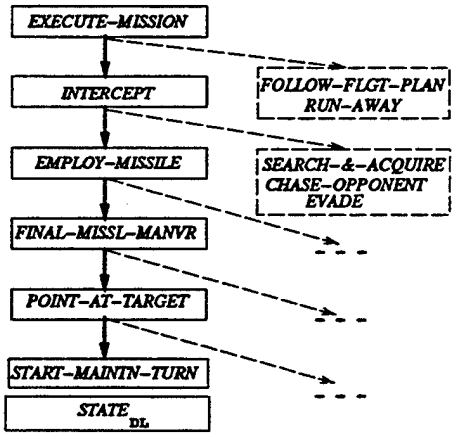
In an environment such as air-combat simulation, agents possess similar behavioral flexibility and reactivity. Thus, the (architectural) mechanisms that an agent employs in generating its own behaviors may be used for tracking others' flexible and reactive behaviors. Consider, for instance, D's tracking of L's behaviors in Figure 1-c. D generates its own behavior using the operator hierarchy shown in Figure 2-a. (The solid lines indicate the actual hierarchy, and the dashed lines indicate unselected options.) Here, at the top-level, D is executing its mission — to protect its home-base for a given time period — via the *execute-mission* operator. Since the termination condition of this operator — completion of D's mission — is not yet achieved, a subgoal is generated.[2] D rejects options such as *follow-flight-plan* and *run-away* in this subgoal in favor of the *intercept* operator, so as to combat L. In service of *intercept*, D selects *employ-missile* in the next subgoal. However, since D has not reached its missile firing range and position, it selects *get-firing-position* in the next subgoal. Skipping to the final subgoal, *maintain-heading* enables D to maintain

[2] A Soar operator has termination conditions — if the operator's application (or new sensor input) changes the state so as to satisfy the termination conditions, then that operator and all of its subgoals are terminated. If the termination conditions remain unsatisfied, a subgoal is created, within which new operators are applied.

its heading, as seen in Figure 1-c.



(a) Operator$_D$ Hierarchy



(b) Operator$_{DL}$ Hierarchy

Figure 2: Operator hierarchies: Solid lines indicate actual selections; dashed indicate unselected options.

The operators used for generating D's own actions, such as in Figure 2-a, will be denoted with the subscript D, e.g., *intercept*$_D$. Operator$_D$ will denote an arbitrary operator of D. State$_D$ will denote the global state shared by all of these operators. It maintains all of the dynamic sensor input regarding D's own aircraft, such as its heading and altitude. It also maintains dynamic radar input regarding L's aircraft, such as heading, range, collision course and other geometric relationships. Additionally, it maintains non-sensor information, e.g., D's missile capabilities. Together, state$_D$ and the operator$_D$ hierarchy constitute the introspectable aspect of D, and in this sense may be considered as D's model of its present self, referred to as model$_D$.

Model$_D$ supports D's flexible/reactive behaviors via its embedding within Soar; and in particular, via two of Soar's architectural features: (i) a decision procedure that supports flexibility by integrating all available knowledge about absolute or relative worth of candidate operators right before deciding to commit to a single operator; (ii) termination conditions for operators that support reactivity by terminating operators in response to the given situation[Rosenbloom *et al.*, 1991]. The point here is not that these specific architectural features are the only way to yield such behavior, but rather that there are such features, and that they can be reused in tracking other agents' behaviors. To illustrate this re-use, we assume for now that D and L possess an identical set of maneuvers. (Note that this sameness of maneuvers is not necessary; all that is required is for D to have an accurate model of its opponent's maneuvers.)

Thus, D uses a hierarchy such as the one in Figure 2-b to track L's behaviors. Here, the hierarchy (the solid lines in Figure 2-b) represents D's model of L's current operators in the situation of Figure 1-c. These operators are denoted with the subscript DL. This operator$_{DL}$ hierarchy, and the state$_{DL}$ that goes with it, constitute D's model of L or model$_{DL}$. Within model$_{DL}$, *execute-mission*$_{DL}$ denotes the operator that D uses to track L's mission execution. Since L's mission is not yet complete, D applies the *intercept*$_{DL}$ operator in the subgoal to track L's intercept. The unselected alternatives here, e.g., *run-away*$_{DL}$, indicate the ambiguity in tracking L's actions (however, assume for now that this is accurately resolved). In the next subgoal, *employ-missile*$_{DL}$ is applied. Since L has reached its missile firing position, in the next two subgoals, *final-missile-maneuver*$_{DL}$ tracks L's final missile maneuver, and *point-at-target*$_{DL}$ tracks L's turning to point at D. In the final subgoal, D applies the *start-&-maintain-turn*$_{DL}$ operator to state$_{DL}$, which does not (can not) actually cause L turn. Instead, this operator predicts L's action and matches the prediction against L's actual action. Thus, if L starts turning to point at D's aircraft, then there is a match with model$_{DL}$'s predictions — D believes L is turning to point at its target, D, to fire a missile. When L's aircraft turns sufficiently to point straight at D's aircraft (Figure 1-c), the termination condition of the *point-at-target*$_{DL}$ operator is satisfied, and it is terminated. A new operator, *push-fire-button*$_{DL}$, is then applied in the subgoal of *final-missile-maneuver*$_{DL}$. This operator predicts a missile firing, although the missile cannot actually be observed. State$_{DL}$ maintains a representation of the missile, and marks it with a low likelihood. Following that, the *fpole-right*$_{DL}$ operator predicts L's right turn for an fpole. When this prediction is matched with L's turn in Figure 1-d, the missile's likelihood is changed to high. D now attempts to evade the missile, with *beam-right*$_D$. (D currently chooses arbitrarily between the execution of operator$_D$ and operator$_{DL}$, as it generates its own actions, while also tracking L's actions.)

The above agent tracking process is related to previous work on *model tracing* in intelligent tutoring systems(ITS) for tracking student actions[Anderson *et al.*, 1990; Ward, 1991]. However, that work has primarily focused on static environments. A recently developed ITS, REACT[Hill and Johnson, 1994], extends model tracing to a more dynamic environment. REACT relies upon a plan-driven tracking strategy, and deals with the more dynamic aspects of the domain as special cases. It specifically abstracts away from tracking students' mental states. In contrast, pilots appear to track their op-

ponents' behaviors in more detail. Such tracking is supported here via a uniform apparatus for the generation of an agent's own flexible/reactive behaviors and tracking other agents' behaviors. In particular, $operator_D$ and $operator_{DL}$ are selected and terminated in the same flexible manner. Thus, as $state_{DL}$ changes, which it does in reflecting the changing world situation, new $operators_{DL}$ may get selected in response. This is key to RESC's situatedness — L's on-going actions are continually tracked in the context of the current $state_{DL}$. For further details on this tracking technique, please see[Tambe and Rosenbloom, 1995].

3 Real-time Ambiguity Resolution

Ambiguity manifests itself in two forms in the agent tracking process introduced in the previous section. One form involves the alternative operators available for tracking the other agent's actions, as seen in the dashed boxes in Figure 2-b. Given these alternatives, it is difficult to make accurate selections of $operator_{DL}$, such that their predictions successfully match L's actions. Should an $operator_{DL}$ selection be inaccurate, in typically results in a *match failure* (if not immediately, then in some further $operator_{DL}$ application). Thus, in Figure 2-b, the $operator_{DL}$ hierarchy predicts L will turn to point at D's aircraft. Suppose this prediction is inaccurate, and L turns in the opposite direction. This difference in the anticipated and actual action leads to a match failure, indicating an inaccuracy in tracking. Similar match failures can also occur if L fails to begin (or stop) turning or maintain heading as anticipated.

A second form of ambiguity is seen in $state_{DL}$. $State_{DL}$ needs to maintain the same types of information as are in $state_D$. Here, there is ambiguity related to both the dynamic sensor information and the static non-sensor information. With respect to static information, there are ambiguities about L's radar and missile capabilities. Even if these are resolved, there are ambiguities about dynamic information, such as whether L has detected D on radar. For instance, in Figure 1-a, based on the static radar range information, D assumes it has arrived within L's radar range; but L may or may not have detected D, depending on its radar's orientation. Such ambiguities in $state_{DL}$ are intimately connected to ambiguities in $operator_{DL}$, since the $operator_{DL}$ hierarchy is dependent on the current $state_{DL}$. Thus, if D assumes it is not detected on L's radar, then the $intercept_{DL}$ operator is ruled out, since there is nothing for L to intercept. In contrast, if D assumes L has detected it, then $intercept_{DL}$ is a likely possibility. A subgoal of $intercept_{DL}$ predicts L's turn to collision course, which is matched by L's turn in Figure 1-a — D now believes L has detected it, and L is going to collision course to intercept. Note that, if D believes that L has detected it, $state_{DL}$ needs to maintain the various dynamic inputs that D believes L obtains from its radar regarding D's heading, range, geometric relationships etc. Fortunately, many of these quantities are symmetric and can be reused from corresponding quantities in $state_D$.

It is difficult to resolve the above ambiguities using methods that have been previously suggested in the model tracing literature. Ward[Ward, 1991] notes that previous model tracing systems have mostly relied on communication with the modeled agent to resolve ambiguities. In air-combat simulation, such communication with enemy pilots is clearly ruled out. Ward's solution in the absence of such information is an exhaustive backtrack search of all of the different alternatives. In the example in 2-b, this involves an attempt to execute and match other operator hierarchies — generated by alternatives such as $run\text{-}away_{DL}$ or $follow\text{-}flight\text{-}plan_{DL}$ — via a systematic backtrack search before committing to $intercept_{DL}$. Unfortunately, this search-then-commit approach will very likely cause tracking to lag far behind the rapidly changing world, precluding D from responding to L's maneuvers in real-time. Furthermore, given the volume of dynamic information on $state_{DL}$, the maintenance of multiple old copies of $state_{DL}$ for backtracking could itself consume non-trivial amounts of space and time. Parallel real-time search of alternative $models_{DL}$ could eliminate the backtracking; however, we will focus on a sequential solution given the implementation technology available to us. Furthermore, parallelism may not be adequate when faced with the expected combinatorics in the number of alternatives. Borrowing ambiguity resolution methods from the plan recognition literature would be yet another possibility; but the computational costs (intractability) of techniques such as automated deduction[Kautz and Allen, 1986] are a significant concern.

So instead, we propose a new approach called RESC (REal-time Situated Commitments) that addresses the above concerns. As seen earlier, RESC's situatedness arises from its tracking of L's on-going actions and behaviors in the context of the current $state_{DL}$. RESC's commitment is to a single $model_{DL}$, with a single $state_{DL}$ that records the on-going world situation in real-time and a single $operator_{DL}$ hierarchy, that provides an on-going interpretation of an opponent's actions. Given the intense real-time pressure, RESC does not spend time trying to match alternatives; instead, it just commits to a single $operator_{DL}$ hierarchy, and any facts inferred in $state_{DL}$ due to this hierarchy. It then tries to use these commitments as context for tracking L's future actions. However, in some cases, the commitments may get withdrawn given RESC's situatedness — as $state_{DL}$ changes, it may satisfy the termination conditions of an $operator_{DL}$ and thus cause it, and all of its subgoals, to terminate.

When faced with ambiguity, it is possible that RESC commits to an inaccurate $operator_{DL}$ and $state_{DL}$, leading to a match failure. RESC recovers from such failures by relying on a method called *single-state backtracking*, that undoes some of its commitments, resulting in the generation of new $operator_{DL}$ hierarchies, in real-time. Of course, if RESC makes more intelligent commitments in the first place — by reducing the ambiguity in the situation with which it is faced — there will be less of a need for undoing its commitments. Subsection 3.1 first describes strategies — some general and some domain specific — used for reducing ambiguities in both $state_{DL}$ and $operator_{DL}$. Subsection 3.2 then

describes single-state backtracking.

3.1 Reducing Ambiguities

There are two classes of strategies used in RESC to resolve ambiguities: *active* and *passive*. The active strategies rely upon an agent's active participation in its environment to gather information to resolve ambiguities. In particular, an automated pilot, such as D, can act in its environment and force its opponent L to react and provide disambiguating information. Consider again the example in Figure 1-a. As discussed earlier, D faces ambiguity in state_{DL} about whether L's radar has detected D. This gets resolved with L's turn to collision course. Unfortunately, if L just happens to be on collision course, it may not turn any further, and the ambiguity would be more difficult to resolve. In such cases, D can randomly turn 15-20°, as shown in Figure 1-b, causing L to react if it wishes to maintain collision course. This provides D sufficient disambiguating information — L's radar has detected D. Unfortunately, D's actions in service of active ambiguity resolution may interfere with its other goals, such as firing a missile at L. In general, such interference is difficult to resolve. Therefore, currently, active ambiguity resolution is based on a fixed set of known maneuvers (supplied by human experts).

In contrast, passive ambiguity resolution strategies rely on existing information to resolve ambiguities. One key piece of information is that in this hostile environment, an opponent is likely to engage in the most harmful maneuver. This information is used in the form of a *worst case* strategy for disambiguation. Thus, given a choice, D always selects the worst-case $\text{operator}_{\text{DL}}$ (from its own perspective) while tracking L's actions. For instance, if there is ambiguity between $\textit{run-away}_{\text{DL}}$ or $\textit{intercept}_{\text{DL}}$, D will select $\textit{intercept}_{\text{DL}}$, which is more harmful. Similarly, D resolves ambiguity in the static information in state_{DL} via the worst-case strategy, e.g., it assumes that L's aircraft is carrying the most powerful missiles and radar that it can carry. Unfortunately, this worst-case strategy can lead to overly pessimistic behavior. In the absolute worst-case, the only option for D is to run away. Therefore, D applies it selectively, typically in cases where it has to disambiguate rapidly, and yet no other means are available. Thus, as seen above, D does not automatically assume detection by L's radar, even though that would be the worst-case assumption.

A second passive ambiguity resolution strategy is *test incorporation*[Bennett and Dietterich, 1986]. The key idea is to generate fewer incorrect alternatives in ambiguous situations. In particular, model_{DL} generates alternative $\text{operators}_{\text{DL}}$ that are tested by matching against L's actual actions. Observations regarding these actions can be used to avoid generating alternatives that are guaranteed to lead to match failures. For instance, in Figure 1-d, $\textit{fpole-right}_{\text{DL}}$ and $\textit{fpole-left}_{\text{DL}}$ are two alternatives available to D in tracking L's actions. If D already sees L turning to its right, then $\textit{fpole-left}_{\text{DL}}$ can be eliminated, since it would be guaranteed to lead to a match failure. Test incorporation relies on such spatial relationships.

A third passive ambiguity resolution strategy is *goal incorporation* (e.g., see [Van Beek and Cohen, 1991]). The key idea here is to resolve ambiguities only to the extent necessitated by an agent's goals. For example, given the reality of the simulation environment, L's aircraft often unintentionally deviates from its intended heading. Given such deviations, L sometimes makes corrections to its headings. However, D does not really need to track and disambiguate these small deviations and corrective actions. It therefore uses *fuzz-box* filters that disregard specified deviations in L's actions. For instance, for $\textit{point-at-target}_{\text{DL}}$, which tracks L's pointing maneuver (Figure 1-c), the fuzz-box filter disregards 5° of deviation in L's heading. Such filtering also helps to avoid tracking of detailed aspects of state_{DL}, and avoids ambiguities there.

3.2 Single-State Backtracking in RESC

Based on the above disambiguation strategies, RESC commits to a single state_{DL} and a single $\text{operator}_{\text{DL}}$ hierarchy, which track L's actions as described in Section 2. However, should this cause a match failure, single-state backtracking is used to undo some commitments. As its name suggests, this backtracking takes place within the context of a single state_{DL}. Starting from the bottom of the $\text{operator}_{\text{DL}}$ hierarchy, operators are terminated one by one in an attempt to get alternatives to take their place. Some alternatives do get installed in the hierarchy, and possibly change state_{DL}, but lead to match failures. These are also replaced, until some alternative leads to an $\text{operator}_{\text{DL}}$ hierarchy that culminates in match success.[3]

Why is this process real-time? The main reason is that backtracking occurs without a re-examination of past sensor input or mental recreation of older $\text{states}_{\text{DL}}$. In particular, while backtrack search would normally involve revisiting old $\text{states}_{\text{DL}}$ and reconsidering the different $\text{operators}_{\text{DL}}$ possible in each of those states — creating an opening for combinatorics — RESC completely avoids such computation. Furthermore, although RESC does backtrack over the operator hierarchy, there are three factors that ameliorate the combinatorics there. First, given RESC's situatedness, backtracking remains tied to the present state_{DL}. Thus, while a match failure is recognized and the backtrack process begun, L and D's aircraft continue to move and turn, changing their speeds, headings, altitudes, and relative geometric relationships (e.g., range, collision course, etc). State_{DL} is continuously updated with this latest information. The backtracking process takes place in the context of this continuously changing state. Thus, only those alternative $\text{operators}_{\text{DL}}$ that are relevant to the current state_{DL} get applied. Similarly, in some cases, changes in state_{DL} cause portions of the $\text{operator}_{\text{DL}}$ hierarchy to terminate automatically during the backtrack process. In other words, RESC is continuously dragged forward as the world changes. Second, RESC does not oblige D to address the match failure before D can execute

[3] In a few cases, there are pending changes related to ambiguities in state_{DL}, e.g., has L detected D? These are applied first, hoping they cause changes to $\text{operator}_{\text{DL}}$ and lead to success.

any of its own operators$_{\mathbf{D}}$. Thus D is free to act to the extent it can. Finally, indeed, if the world were to magically become static, RESC's strategy will result in a complex search, although still within the context of a single state$_{\mathbf{DL}}$. However, it is unclear if this is necessarily problematic — a static world should possibly merit a more thorough search.

Let us consider some examples of single-state backtracking. As a simple example, suppose D has commited to the model$_{\mathbf{DL}}$ in Figure 2-b. Initially, *point-at-target*$_{\mathbf{DL}}$ has match success in that, as predicted, L indeed starts turning towards D(see Figure 3-a for an illustration). However, L really has decided to run away; so it continues turning 180° without stopping when pointing at D (Figure 3-b). This leads to a match failure in the operator$_{\mathbf{DL}}$ hierarchy. Single-state backtracking now ensues, terminating operators beginning from the bottom of the hierarchy. Finally, *intercept*$_{\mathbf{DL}}$ is terminated and replaced by *run-away*$_{\mathbf{DL}}$. This predicts L to be turning towards its home-base, which successfully matches L's actions (Figure 3-c). Thus, D successfully applies *run-away*$_{\mathbf{DL}}$, predicting and matching L's actions, without mentally recreating the state$_{\mathbf{DL}}$ in which L may have initiated its run-away maneuver.

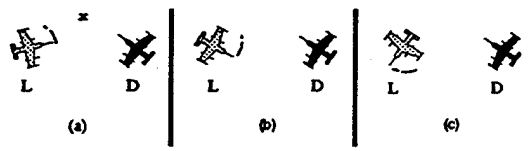


Figure 3: L continues to turn to run away.

A slightly more complex example involves situations where L is engaging in a beam maneuver. Here, D initially matches *fpole-right*$_{\mathbf{DL}}$, and even infers L's missile firing, as part of state$_{\mathbf{DL}}$. However, as L keeps turning, there is soon a match failure, causing D to backtrack until *beam-right*$_{\mathbf{DL}}$ successfully matches. There are two key points here. First, again D is successful in applying *beam-right*$_{\mathbf{DL}}$, without mentally recreating the state$_{\mathbf{DL}}$ in which L may have initiated its beam maneuver. Second, D's earlier inference of L's missile firing is not removed, even though it is based on a sequence of operators that eventually led to a match failure. This is because it is difficult for D to decide if L was initially maneuvering to fire a missile and then switched to beam, or if it was always engaged in beam. Not knowing any better, D does not eliminate the earlier inference from state$_{\mathbf{DL}}$. Fortunately, when aircraft turn 90° to beam, they cannot provide radar guidance to their missiles. Therefore, with L's beam, D infers that the missile that it earlier inferred on state$_{\mathbf{DL}}$ has lost guidance and become harmless. The end result is identical to a case where D had successfully tracked L's beam maneuver, without the failed intermediate inference of an *fpole-right*$_{\mathbf{DL}}$ maneuver.

We have so far found RESC's single-state backtracking to be successful in the air-combat simulation domain (see Section 4). Given the potential application of this approach for other areas of real-time comprehension, it is useful to analyze the reasons behind its success. Towards this end, consider first the following simplified and abstract characterization of a successful application of single-state backtracking: L initiates some maneuver β at time *T0*. However, at *T0*, D attempts to match it by applying an operator $\alpha_{\mathbf{DL}}$ to state$^{T0}_{\mathbf{DL}}$ (which denotes state$_{\mathbf{DL}}$ at time *T0*). At time *T0*+τ, in state$^{T0+\tau}_{\mathbf{DL}}$ D recognizes a match failure with $\alpha_{\mathbf{DL}}$. It backtracks and applies $\beta_{\mathbf{DL}}$ to state$^{T0+\tau}_{\mathbf{DL}}$. The key observation here is that despite the time delay τ and the intervening application of $\alpha_{\mathbf{DL}}$, $\beta_{\mathbf{DL}}$ is successful in predicting and matching L's maneuvers, as though it were applied to state$^{T0}_{\mathbf{DL}}$. Based on this observation, in terms of operator preconditions and effects, we can infer at least three requirements that need to be met for single-state backtracking to work. In the following, we list these requirements, and illustrate how pilot agents currently adhere to them:

1. *The preconditions of* $\beta_{\mathbf{DL}}$ *are satisfied in* state$^{T0+\tau}_{\mathbf{DL}}$ *in much the same way as in* state$^{T0}_{\mathbf{DL}}$: For pilot agents, operator preconditions are expressed so they do not test specific positions of aircraft, but rather abstracted features of state$_{\mathbf{DL}}$ — similar to the fuzz-boxes in Section 3.1 — that are unlikely to change in τ. That is, abstracted features tested by preconditions change at a rate smaller than $1/\tau$.
2. *The effects of* $\beta_{\mathbf{DL}}$ *when applied to* state$^{T0+\tau}_{\mathbf{DL}}$ *are equivalent to the effects of* $\beta_{\mathbf{DL}}$ *when applied to* state$^{T0}_{\mathbf{DL}}$: This is achieved by expressing operator effects relative to some feature of state$_{\mathbf{DL}}$ that is unlikely to have changed in the intervening time period τ. For instance, the effect of *run-away*$_{\mathbf{DL}}$ predicts L is headed towards its home base — the location of this base is unlikely to change within τ. Similarly, the effects of operators such as *beam-right*$_{\mathbf{DL}}$ are expressed in relative terms as L turning to achieve 90° angle-off, which is an angle formed by L's heading relative to the straight line joining D and L (while this line does change its position in τ, given the range between D and L the change is small, and gets absorbed by the fuzz-boxes). If the effects were expressed instead as turning 90° from L's current heading, they would have provided very different results at *T0* and *T0*+τ, defeating single-state backtracking. Overall, the above seems possible because operators in this environment typically strive to achieve positions relative to slowly changing reference points, such as turning to a particular heading relative to an opponent's aircraft, or relative to a waypoint such as the home-base.
3. *The effects of* $\alpha_{\mathbf{DL}}$ *as applied to* state$^{T0}_{\mathbf{DL}}$ *are eliminated at some time* $T0+\tau$ *before they cause inconsistencies in* D*'s response*: As seen in an example above, even though L's missile firing was inferred, and this inference was not "cleaned up" upon recognition of a match failure, L's future maneuver automatically nullifies the effect of that inference. Fortunately, typical operator$_{\mathbf{DL}}$ applications do not

commit to such inferences on $state_{DL}$. For those that do commit, these commitments get removed by future maneuvers or become irrelevant.

In some cases, L quickly terminates its maneuver β within time period τ, and initiates a new one γ at time $T0+\tau$. Here, given RESC's situatedness, at time $T0+\tau$, D completely skips tracking β_{DL}, and tracks γ_{DL} instead. Fortunately, since D's initial attempt is to apply worst-case $operators_{DL}$, there is at least the assurance that what is skipped (β_{DL}) is not among the worst of the possibilities.

4 Implementation and Evaluation

To understand the effectiveness of the RESC approach, we have implemented it as part of an experimental variant of TacAir-Soar[Tambe *et al.*, 1995]. The current TacAir-Soar system contains about 2000 rules, and automated pilots based on it have participated in combat exercises with expert human pilots. Our experimental version — created to investigate the RESC approach — contains about 950 of these rules. Proven techniques from this experimental version, called TacAir-Soar^{RESC} are transferred back into the original TacAir-Soar.

There are at least two aspects to understanding the effectiveness of TacAir-Soar^{RESC}. The first aspect is whether the current approach enables D, the TacAir-Soar^{RESC} pilot agent, to track its opponents' actions accurately in real-time. We conducted two sets of experiments to address this issue. The first set involved running Soar-vs-Soar air-combat simulation scenarios (as outlined by the human experts). The results from these experiments are presented in Table 1.

Scn. num	Num oppnts	Total opertrs	% operators agent track	% of colm 2 in match fail
1	1	37	8%	0%
2	1	133	45%	17%
3	2	167	50%	16%
4	2	175	54%	17%
5	4	142	63%	11%

Table 1: Results of Soar-vs-Soar experiments.

The first column lists the scenario number. The second column lists the number of opponents that D faces in each scenario — this varies from one to four in these scenarios. The third column indicates the total number of D's operator executions in each scenario. This includes operators for D's own actions, as well as for tracking opponent actions. The total number provides some indication of the comparative complexity of the different scenarios. Note that these operators are not all applied in a sequence at regular intervals; D often waits in between applications as it tries to get into different positions. Indeed, despite the differences in the number of operators, the total time per scenario is about the same, approx 5 minutes. The fourth column shows the percentage of operator executions involved in agent tracking. This percentage clearly depends on the number of maneuvers that the opponents execute, and the number of opponents. The key point here is that agent tracking is a non-trivial task for D. Furthermore, higher percentages of operator executions may be dedicated to agent tracking with increased numbers of opponents.

The fifth column shows the percentage of agent tracking operators involved in match failures (counting operators at the bottom of the hierarchy that encountered the failure, but not their parents). The main point here is that the overall percentage of these operator is low; at most 17% of the agent tracking operators are involved in match failures.

In all of these cases, D is successful in tracking opponents in real-time so as to react appropriately. Even in cases where D encounters match failures, it is able to backtrack to track the on-going activities in real-time and respond appropriately. However, as the number of opponents increases, D does face resource contention problems. With four opponents, it is unable to track the actions of all of the agents in time, and gets shot down (hence fewer operators). This resource contention issue is under active investigation[Tambe, 1995].

Our second set of experiments involved Soar-vs-ModSAF simulated air-combat scenarios. ModSAF-based[Calder *et al.*, 1993] pilot agents are controlled by finite state machines combined with arbitrary pieces of code, and do not exhibit high behavioral flexibility. While D was in general successful in agent tracking in these experiments — it did recognize the maneuvers in real-time and respond to them — one interesting issue did come up. In particular, in one of the scenarios here, there was a substantial mismatch in D's worst assumptions regarding its opponent's missile capabilities and the actual capabilities — leading to tracking failures. Dealing with model mismatch is also an issue for future work.

The second aspect to understanding the effectiveness of TacAir-Soar^{RESC} is some quantitative estimate of the impact of agent tracking on improving D's overall performance. In general, this is a difficult issue to address (see for instance the debate in [Hanks *et al.*, 1993]). Nonetheless, we can at least list some of the types of benefits that D accrues from this capability. First, agent tracking is crucial for D's survival. Indeed, it is based on agent tracking that D can recognize an opponent's missile firing behavior and evade it. Second, agent tracking improves D's overall understanding of a situation, so it can act/react more intelligently. For instance, if an opponent is understood to be running away, D can chase it down, which would be inappropriate if the opponent is not really running away. Similarly, if D is about to fire a missile, and it recognizes that the opponent is also about to do the same, then it can be more tolerant of small errors in its own missile firing position so that it can fire first. Finally, agent tracking helps D in providing a better explanation of its behaviors to human experts. (Such an explanation capability is currently being developed[Johnson, 1994]). If human experts see D as performing its task with an inaccurate understanding of opponents' actions, they will not have sufficient confidence to actually use it in training.

5 Lessons Learned

This paper presented an approach called RESC, for agent tracking in real-time dynamic environments. Our investigation was based on a real-world synthetic environment that has already been used in a large-scale operational military exercise[Tambe *et al.*, 1995]. Lessons learned from this investigation — as embodied in RESC — are as follows:

- *To track other agents' flexible and reactive behaviors*: Reuse the architectural mechanisms that support an agent's own flexible/reactive behaviors in service of tracking others' behaviors.
- *To address ambiguities in real-time*: Quickly commit to a single interpretation, and use single-state backtracking to recover from erroneous commitments.
- *To address real-time issues in general*: Keep tracking firmly tied to the *now*, i.e., to the present state.

One key issue for future work is investigating the generality of these lessons by applying RESC to other competitive and collaborative multi-agent domains. One candidate that has been suggested is a real-time multi-robot domain where robots track other robots or humans to collaborate in a task by observation (rather than by communication)[Kuniyoshi *et al.*, 1994]. Beyond agent tracking, there is some indication that RESC could apply in other real-time comprehension tasks. For instance, a RESC-type strategy has been previously used in a real-time language comprehension system[Lewis, 1993]. This system also commits to a single interpretation of an input sentence despite ambiguity, and attempts to repair the interpretation in real-time when faced with parsing difficulties. We hope that investigating these broader applications will lead to an improved understanding of agent tracking and comprehension.

References

[Anderson *et al.*, 1990] J. R. Anderson, C. F. Boyle, A. T. Corbett, and M. W. Lewis. Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, 42:7–49, 1990.

[Azarewicz *et al.*, 1986] J. Azarewicz, G. Fala, R. Fink, and C. Heithecker. Plan recognition for airborne tactical decision making. In *Proceedings of the National Conference on Artificial Intelligence*, pages 805–811. Menlo Park, Calif.: AAAI press, 1986.

[Bennett and Dietterich, 1986] J. S. Bennett and T. G. Dietterich. The test incorporation hypothesis and the weak methods. Technical Report 86-30-4, Department of Computer Science, Oregon State University, 1986.

[Calder *et al.*, 1993] R. B. Calder, J. E. Smith, A. J. Courtemanche, J. M. F. Mar, and A. Z. Ceranowicz. Modsaf behavior simulation and control. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*, 1993.

[Hanks *et al.*, 1993] S. Hanks, M. E. Pollack, and P. R. Cohen. Benchmarks, test beds, controlled experimentation, and the design of agent architectures. *AI Magazine*, 14(4):17–42, 1993.

[Hill and Johnson, 1994] R. Hill and W. L. Johnson. Situated plan attribution for intelligent tutoring. In *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, Calif.: AAAI press, 1994.

[Johnson, 1994] W. L. Johnson. Agents that learn to explain themselves. In *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, August 1994. Menlo Park, Calif.: AAAI press.

[Kautz and Allen, 1986] A. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of the National Conference on Artificial Intelligence*, pages 32–37. Menlo Park, Calif.: AAAI press, 1986.

[Kuniyoshi *et al.*, 1994] Y. Kuniyoshi, S. Rougeaux, M. Ishii, N. Kita, S. Sakane, and M. Kakikura. Cooperation by observation – the framework and the basic task pattern. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1994.

[Lewis, 1993] R. L. Lewis. An architecturally-based theory of human sentence comprehension. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, 1993.

[Newell, 1990] A. Newell. *Unified Theories of Cognition.* Harvard Univ. Press, Cambridge, Mass., 1990.

[Rich and Knight, 1990] E. Rich and K. Knight. *Artificial Intelligence.* McGraw-Hill, New York, NY, 1990.

[Rosenbloom *et al.*, 1991] P. S. Rosenbloom, J. E. Laird, A. Newell, , and R. McCarl. A preliminary analysis of the soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47(1-3):289–325, 1991.

[Tambe and Rosenbloom, 1995]
M. Tambe and P. S. Rosenbloom. Event tracking in a dynamic multi-agent environment. *Computational Intelligence*, (To appear), 1995. WWW: http://www.isi.edu/soar/tambe/event.html.

[Tambe *et al.*, 1995]
M. Tambe, W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), Spring 1995.

[Tambe, 1995] M. Tambe. Recursive agent and agent-group tracking in a real-time dynamic environment. In *Proceedings of the International Conference on Multi-agent systems (ICMAS)*, June 1995.

[Van Beek and Cohen, 1991] P. Van Beek and R. Cohen. Resolving plan ambiguity for cooperative response generation. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 938–944, 1991.

[Ward, 1991] B. Ward. *ET-Soar: Toward an ITS for Theory-Based Representations.* PhD thesis, School of Computer Science, Carnegie Mellon Univ., 1991.

Building Intelligent Pilots for Simulated Rotary Wing Aircraft

Milind Tambe, Karl Schwamb and Paul S. Rosenbloom
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
email: {tambe, schwamb, rosenbloom}@isi.edu

1. Abstract

The Soar/IFOR project has been developing intelligent pilot agents (henceforth IPs) for participation in simulated battlefield environments. While previously the project was mainly focused on IPs for fixed-wing aircraft (FWA), more recently, the project has also started developing IPs for rotary-wing aircraft (RWA). This paper presents a preliminary report on the development of IPs for RWA. It focuses on two important issues that arise in this development. The first is a requirement for reasoning about the terrain — when compared to an FWA IP, an RWA IP needs to fly much closer to the terrain and in general take advantage of the terrain for cover and concealment. The second issue relates to code and concept sharing between the FWA and RWA IPs. While sharing promises to cut down the development time for RWA IPs by taking advantage of our previous work for the FWA, it is not straightforward. The paper discusses the two issues in some detail and presents our initial resolutions of these issues.

2. Introduction

The Soar/IFOR project has been developing intelligent pilot agents (IPs) for simulated battlefield environments (Laird et al., 1995, Rosenbloom, et al., 1994, Tambe et al., 1995). Until Summer 1994, the project was focused on building IPs for simulated fixed-wing aircraft (FWA), including air-to-air fighters and ground-attack aircraft. Since July 1994, we have begun developing IPs for simulated rotary-wing aircraft (RWA), specifically, AH-64 Apache attack helicopters.

While there are similarities in an RWA and an FWA pilot's missions — e.g., employing weapons on targets, flying mission-specified routes — there are also some important differences. One key difference is reasoning about the terrain. For example, an RWA pilot's mission can involve flying Nap-of-the-earth (NOE), where it needs to fly only about 25 feet above ground level, while avoiding obstacles. It may also involve flying through a valley, or around a forested region. The mission may also involve hiding (masking) behind a ridge, popping up to spot enemy targets, and remasking in a new hiding position. Figure 1 provides an illustration of this type of terrain reasoning. It presents a snapshot, taken from ModSAF's plan-view display (Calder et al., 1993), of a typical scenario involving Soar-based RWA IPs. There are two RWA in the scenario, just behind the ridge, indicated by the contour lines. The other vehicles in the figure are a convoy of "enemy" ground vehicles — tanks and anti-aircraft vehicles — controlled by ModSAF. The RWA are approximately 2.5 miles from the convoy. The IPs have hidden their helicopters behind the ridge (their approximate hiding area is specified to them in advance). They unmask these helicopters by popping out from behind the ridge to launch missiles at the enemy vehicles, and quickly remask (hide) by dipping behind the ridge to survive retaliatory attacks. They subsequently change their hiding position to avoid predictability when they pop out later.

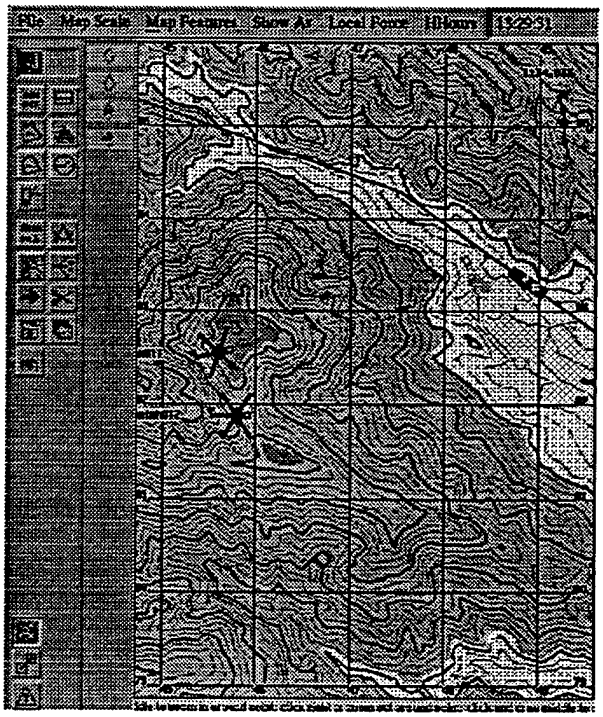


Figure 1: A snapshot of ModSAF's simulation of an air-to-ground combat situation.

Thus, the development of RWA IPs brings up the novel issue of terrain reasoning, not addressed in previous work on Soar/IFOR agents. There has been much work on terrain reasoning in ModSAF in their development of semi-automated forces or SAFs

(Calder et al., 1993). That work has so far primarily focused on ground-based SAFs (e.g., (Longtin, 1994)), although there is a recent effort focused on terrain reasoning for RWA (Tan, 1995). Outside the arena of automated forces, terrain reasoning in the form of route planning and execution has been addressed extensively in AI and Robotics. The focus of much of this work is on 2D routes (Denton and Froeberg, 1984, Khatib, 1986, Lozano-Perez and Wesley, 1979, Mitchell, 1990) — and this category includes some previous work within Soar (Stobie et al., 1992) — although some efforts have also attacked the 3D route planning problem (Bose et al., 1987, Rao and Arkin, 1989). Other aspects of terrain reasoning such as tactical situation assessment (McDermott and Gelsey, 1987) and hiding (Stobie et al., 1992) have also received some attention, although not nearly as much as route planning. As discussed in Section 3, the pure route planning approaches from this literature are unlikely to address the terrain reasoning challenge facing the RWA IPs, which is to accomplish these tasks in real-time, given a realistic 3D terrain database. A hybrid solution combining some abstract plans with reactivity is currently being investigated.

Given the similarities between the FWA and RWA IPs, concept and code sharing between the two is a real possibility. Sharing would speed up development of RWA IPs by taking advantage of our previous work on FWA. However, the differences — such as the terrain reasoning capability above — imply that sharing is not straightforward. There have been some previous efforts aimed at facilitating reuse of code and concepts among Soar systems. These efforts have typically focused on reuse of individual capabilities, such as inductive learning (Rosenbloom and Aasman, 1990), or natural language (Lewis, 1993, Rubinoff and Lehman, 1994) capabilities. The novel issue here is that a large fraction of the FWA IP structure is potentially reusable in developing RWA IPs and such reuse needs to be facilitated.

The rest of this paper provides more details on these two issues. Section 3 focuses on terrain reasoning. Section 4 discusses the issue of code and concept sharing between Soar-based FWA and RWA IPs. We will assume some familiarity with the Soar architecture (Laird, Newell, and Rosenbloom, 1987, Rosenbloom, et al., 1991).

3. Terrain Reasoning

The overall terrain reasoning tasks for an RWA IP may be subdivided into two categories. The first is to fly from a given source to a destination, while abiding by mission specified constraints regarding the flight methods. A flight method primarily specifies maintenance of a certain air-speed and altitude above ground level. In particular, a *high-level* flight requires that the RWA fly more than 200 feet above ground level with air-speed as high as 145 knots. A *low-level* flight requires that the RWA fly 100-200 feet above ground level, while maintaining a maximum air-speed of 100 knots. A *contour* flight requires the RWA to fly between 25-100 feet above ground level, but with a maximum air-speed of 70 knots. An *NOE* flight requires the RWA to fly within just 25 feet above ground level, with a maximum air-speed of 40 knots. Additionally, an NOE flight may require that an RWA fly through a valley along a hillside, or through a narrow clear corridor in a forested region. The second category of terrain reasoning tasks involves an RWA IP's activities once it successfully follows its route to its battle area, and possibly engages enemy vehicles. Its activities in this area involve selecting and occupying good hiding positions (behind a ridge or a forested region) and flying between hiding positions while remaining concealed from a possibly mobile enemy. It may also involve reasoning about possible enemy hiding positions.

For both categories of tasks, one key issue for an RWA IP is to execute them in the context of a large-scale and realistic 3D terrain database, with features such as rivers, ridges, valleys, hills and forested regions. A second key issue is that given its complexity, the cost of sensing and processing large tracts of the terrain database is non-trivial. A third related issue is that an IP has to exhibit human-like behavior in performing these terrain reasoning tasks. Thus, it should not make use of information that a human pilot is unlikely to obtain. For example, as with a human pilot, an IP should plan routes using a map of the terrain database (which possibly may be inaccurate), rather than using the actual terrain database (which would always be 100% accurate). A final issue is that an IP has to perform its tasks in real-time. The following two subsections illustrate how these issues are addressed for each of the two types of tasks above.

3.1. Route Flying

For the task of route flying, one possible approach for addressing the above issues would be to use one of a variety of path-planning methods that provides a very detailed 3D point-to-point route, with little need or freedom to modify the given route (Stobie et al., 1992, Bose et al., 1987, Rao and Arkin, 1989, Denton and Froeberg, 1984). One such approach, based on weighted-region path planning (Mitchell, 1990), is to conceptually divide a map of the terrain into 3D cells (cubes), assign an appropriate cost to each cell that reflects mission-specified constraints, and then search for a minimum cost path through the cells. One advantage of such an approach is that an RWA IP need not sense the terrain database in any detail, but rather just enough to follow its plan. In addition, the

low sensing overhead would facilitate real-time task performance. However, there are several problems with such an approach. First, given the complexity of the terrain, this approach would require a significant initial computational effort to create and then search the cells. Second, it could be wasteful given the realism of the RWA model and its flight controls — it will not be possible for a Soar-based IP to precisely control an RWA to follow such a detailed route, and it will end up having to reactively improvise the path or replan. The original planner could potentially take these realistic flight controls into account when developing a plan — so that no on-line replanning may be required — but that would further increase the complexity of planning. Third, if the map of the terrain is inaccurate or incomplete, the plan generated could be inaccurate as well. Even if the map were completely accurate (or if the IP were using the terrain database itself rather than a map), there could still be deviations from the planned route caused by an unexpected encounter with hostile or friendly vehicles. Thus, an IP may not be able to rely on just its original planned route; it may need to replan. Finally, human pilots typically do not rely on such detailed plans; and thus in forcing IPs to follow such plans, we are likely deviating from our goal of building human-like IPs.

So instead, a Soar-based IP follows a hybrid strategy that combines a plan-based and reactive strategy. In particular, it relies on more abstract route plans, that provide it just two to three intermediate points.[1] The IP then executes these route plans while reacting to sensory information that enables it to abide by the mission specified constraints. For ideal human-like IPs, this sensory information should be precisely what a human pilot would obtain visually by looking out the window. Unfortunately, for an IP, such visual processing is likely to be extremely complex and expensive. Therefore, special inexpensive sensors have been designed that approximate such visual processing. One such sensor is the *look-ahead altitude sensor* or *LAS* sensor. LAS is slaved to the parameters supplied by the IP. The IP sets parameters for LAS that specify a lookahead range and orientation, which in turn specifies a line segment of specific length and orientation originating from the IP's current location. Once these parameters are set, LAS scans the terrain database repeatedly (in fact, each time ModSAF schedules the agent for execution), and returns the highest altitude value along the specified line segment. For instance, to fly NOE, an IP sets LAS's parameters to a lookahead range of 50 meters, and orientation in the direction of its flight. The pilot reacts to LAS's response by modifying the altitude of its helicopter to be approximately 25 feet above the highest point.[2]

The top half of Figure 2 shows a pilot agent making use of LAS to fly NOE. The shaded portion in the figure is a profile of the terrain, while the dashed line is a profile of the helicopter flying NOE. The straight lines indicate LAS's lookahead range while scanning the database. The bottom half of Figure 2 indicates a longer lookahead range, and change in the flight profile that that results.

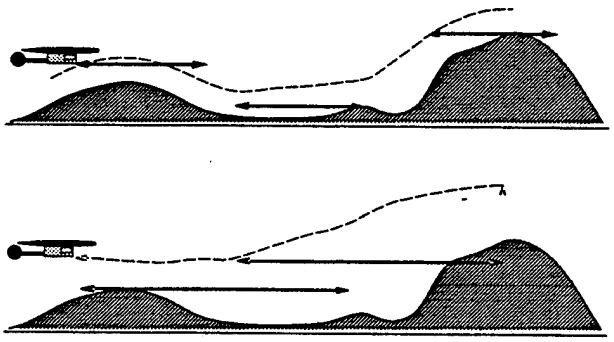


Figure 2: Illustrations of lookahead altitude sensor. LAS scans the terrain database each time the agent is scheduled for execution (illustrations not from an actual run).

The precise value of the lookahead range is determined to a large extent by the speed of the RWA. In particular, for an NOE flight, an IP currently flies conservatively at a speed of 20 knots. With 50 meters lookahead, that gives it about 5 seconds to change its altitude. The other flight methods, specifically contour, low-level and high-level flight, require that the RWA fly at a higher speed. This in turn requires that the IP set a longer lookahead range to give itself more time to react. Speed is however not the only factor determining the lookahead range. It is also dependent on the type of flight profile desired. For instance, at its speed of 80 knots, an IP could potentially sustain the altitude required for its low-level flight with a lookahead of just 200-300 meters. However, the flight profile generated follows the terrain much too closely — it is not as smooth as the flight profile that results from a human pilot's low-level flight (at least as indicated by the experts). Therefore, the low-level flight uses a much longer lookahead range of 1500 meters. The high-level flight uses a lookahead range of 5000 meters.

Unfortunately, long lookahead ranges in LAS could potentially hinder an IP's real-time performance. Therefore, to lower its cost, LAS samples precisely 100 points along the specified line

[1] At present, these abstract routes are provided by a human; although given that they are abstract, planning these routes is expected to be much less complex.

[2] RWA agents in ModSAF appear to follow a similar technique (Tan, 1995).

segment irrespective of the lookahead range. Thus, despite the variation in the lookahead range in Figure 2, LAS will scan precisely 100 points. This sampling resolution may appear to be very low, with the potential of missing high altitude cliffs. However, LAS's repeated scanning in effect improves its sampling resolution. In particular, since an RWA progresses towards its destination between two scans, successive scans sample slightly different points. In fact, each successive scan samples 99 points in the neighborhood of the points from its previous scan (on the same line segment), and one new point. This resolution could still be insufficient for some types terrain. For instance, if the terrain is an urban landscape with a sparse population of pin-shaped high-altitude structures,[3] there is a small possibility that LAS may miss those in its scanning. In such cases, there may be a need to increase the sampling resolution. However, the 100 point scans have so far proved adequate over the terrain database used in our experiments (the RWA have not crashed).

Figure 3 presents a flight profile from an actual run of a Soar-based RWA using the contour flight method. Figure 4 presents a flight profile from another run of a Soar-based RWA over approximately the same terrain, but using the NOE flight method. The shaded portion indicates the terrain, while the dashed line indicates the actual flight profile. IPs smoothen out the flight by using fuzz-boxes (McDermott and Davis, 1984) to avoid excessive altitude adjustments.

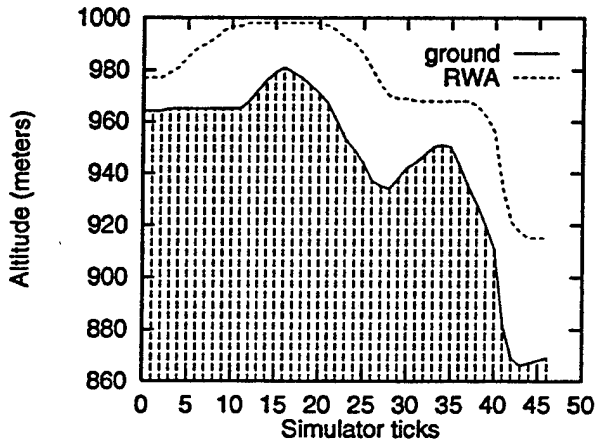


Figure 3: Illustration of a contour flight from an actual run.

Similar low-cost, LAS-type sensors approximating a human pilot's visual input are currently being designed to enable the RWA pilots to fly through valleys.

[3]A clock tower would be one example of such a structure.

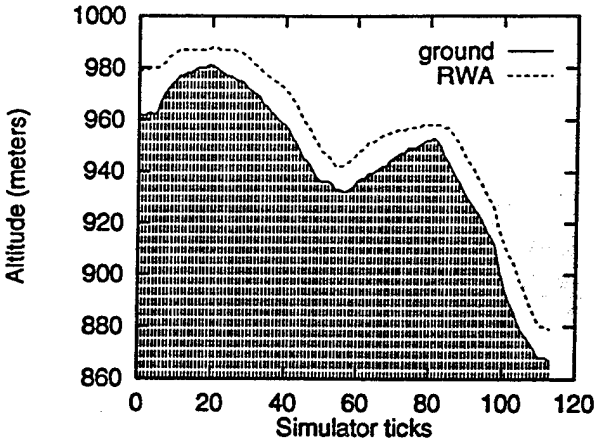


Figure 4: Illustration of an NOE flight from an actual run.

3.2. Hiding

Once an RWA IP reaches its mission-specified battle area, it needs to engage in hiding-related tasks. In general, a battle area could be of an arbitrary (convex) shape, or specified in terms of landmarks, such as trees or rocks. The IP should be capable of locating good hiding positions within this area and move between hiding positions while remaining concealed from its enemy. This second terrain reasoning capability, at least at this level of generality, is very much an issue for future research. At present, we have restricted the battle area to be a rectangle. One side of this rectangular area, typically coinciding with a ridge or a tree line, is a mission specified line segment. This is in essence considered to be an imaginary wall, and any movement behind it is assumed to be hidden from the enemy. An RWA IP hides in a small rectangular area (defined with a width of 100 meters) behind this imaginary wall. When relocating to a new hiding position, it uses the NOE flight method to remain at a low altitude and thus hidden behind the wall. The approximations of a wall and a rectangular area for hiding are both based on our previous work in the *groundworld* domain. Groundworld involved a simulated terrain with random configurations of horizontal and vertical walls, where an intelligent agent had to hide behind a wall to escape from another agent pursuing it (Stobie et al., 1992, Tambe and Rosenbloom, 1993).

4. Sharing and Reuse

RWA pilots' missions have some requirements — such as, identifying enemy vehicles, firing missiles at target vehicles and flying in formation — in common with those of FWA pilots. These commonalities may be exploited to cut down development time by sharing or reusing both code and concepts from Soar-based FWA pilots in the development of RWA pilots. For instance, for an FWA IP, the code for firing a

missile involves three operators that orient its aircraft towards its target, then push a fire button to actually launch the missile, and then guide the missile (should the missile require guidance) via radar (or other) illumination of the target. These three operators can be reused in an RWA IP. At present, a Soar-based RWA IP has 44 operators, with 25 (that is about 57%) reused in some form from the Soar-based FWA IPs. The 19 new operators are those involved with terrain reasoning tasks such as flying NOE, masking and unmasking. This sharing is accomplished simply by loading in appropriate operators from an FWA IP code in an RWA IP.

Differences in concepts and terminology, however, make some of the sharing problematic. For example, for FWA pilots engaged in air-to-air missions, the concept of launch-acceptability-region or LAR of a missile combines both the range to a target and the target aspect (angle between the target's current heading and the straight line joining the target and the FWA pilot's current locations). Thus, if a target is heading towards the FWA pilot with a 0° target aspect, the missile may be fired from a long range; but the range is reduced substantially if the target has a 180° target aspect. In contrast, for an RWA pilot, the target aspect is irrelevant in calculating a missile's LAR — the missile may be fired at an equally long range irrespective of the target aspect. This creates a significant difference in the concept of a missile LAR for an FWA and an RWA IP, making the sharing of missile-LAR-related code difficult. There is an accompanying difference in the terminology as well — the RWA pilot refers to the missile LAR as a missile constraint.

At least some of these apparent discrepancies in the two IP's concepts — and potentially their terminology — could be resolved if the agents reason about the concepts from first principles. For instance, agents could calculate a missile's LAR from first principles, based on the relative velocities (speed and direction) of the missile and the target. Since an FWA IP's target in air-to-air combat is a fighter jet, moving at a speed that may be only a half to a fifth its missile speed, its angle of movement (target aspect) becomes an important factor in calculating LAR. In particular, a target moving towards the FWA allows a missile to be fired from a much longer range; while a target that is moving away requires that the missile be fired from a much closer range, so that the missile may catch up with the target before expending all its fuel. In contrast, an RWA IP's target is moving two orders of magnitude slower than its missile — the angle of the target's movement has a negligible impact on the missile range. In other words, with the first principles calculations, the target aspect discrepancy automatically disappears. It will appear important in FWA IP's calculations, and negligible in an RWA IP's calculations.

While such calculations from first principles would facilitate sharing, the calculations themselves may be prohibitively expensive, and hinder real-time performance. Soar's chunking (learning), could potentially compile such first principles calculations into new rules and alleviate this cost. However, that remains an issue for future work. We are currently relying on a lower cost alternative, where a problematic aspect of the agent code is rewritten when in reuse.

5. Current Status and Future Work

As of February 1995, the RWA agents are capable of performing a complete *attrit* mission, which involves flying to a battle area using one of the possible flight methods, followed by masking, unmasking, firing missiles at targets, and relocating to a different masking location between missile firings. We have run scenarios with up to four RWA IPs executing the *attrit* mission.

At present the RWA IPs can fly in coordination, in pairs. Extending this work to enable coordinated mission execution involving a platoon or a company of RWA agents (with a platoon and a company commander), is at the top of our agenda for future work. Agents at higher echelons of command, such as a company commander, may also bring up issues of communication and mission planning, which we have currently not addressed. Other issues for future work, mentioned in previous sections, include improvement in terrain reasoning for hiding, and in code/concept sharing among Soar agents.

6. Acknowledgements

This research was supported under subcontract to the University of Southern California Information Sciences Institute from the University of Michigan, as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Research Laboratory (NRL); and under contract N66001-95-C-6013 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Command and Ocean Surveillance Center, RDT&E division (NRAD). Critical expertise and support has been provided by David Sullivan of BMH Inc.

7. References

Bose, P. K., Meng, A. C-C., Rajnikanth, M. (1987) Planning flight paths in dynamic situations with incomplete knowledge. Proceedings of the SPIE conference on Spatial reasoning and multi-sensor fusion.

Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., Ceranowicz, A. Z. (1993) ModSAF behavior simulation and control. Proceedings of the Conference on Computer Generated Forces and Behavioral Representation.

Denton, R. V., and Froeberg, P. L. (1984) Applications of Artificial Intelligence in Automated Route Planning. Proceedings of SPIE conference on applications of Artificial Intelligence. , pp. 126-132.

Khatib, O. (1986) "Real-time obstacle avoidance for manipulators and mobile robots". *International Journal of Robotics Research 5*, 1 , 90-98.

Laird, J. E., Johnson, W. L., Jones, R. M., Koss, F., Lehman, J. F., Nielsen, P. E., Rosenbloom, P. S., Rubinoff, R., Schwamb, K., Tambe, M., van Lent, M., and Wray, R., (May, 1995) Simulated Intelligent Forces for Air: The Soar/IFOR project 1995. Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation.

Laird, J. E., Newell, A. and Rosenbloom, P. S. (1987) "Soar: An architecture for general intelligence". *Artificial Intelligence 33*, 1 , 1-64.

Lewis, R. L. (1993) An architecturally-based theory of human sentence comprehension. Proceedings of the Eleventh Annual Conference of the Cognitive Science Society.

Longtin, M. J. (1994) Cover and concealment in ModSAF. Proceedings of the Conference on Computer Generated Forces and Behavioral Representation.

Lozano-Perez, T. and Wesley M. A. (1979) "An algorithm for planning collision-free paths among polyhedral obstacles". *Communications of the ACM 22*, 10 , 560-570.

McDermott, D. and Davis, E. (1984) "Planning routes through uncertain territory". *Artificial Intelligence 22* , 107-156.

McDermott, D., and Gelsey, A. (1987) Terrain analysis for tactical situation assessment. Proceedings of the SPIE conference on Spatial reasoning and multi-sensor fusion.

Mitchell, J. S. B. (1990) Algorithmic approaches to optimal route planning. Proceedings of the SPIE conference on Mobile Robots.

Rao, T. M., and Arkin, R. C. (1989) 3D Path planning for flying/crawing robots. Proceedings of the SPIE conference on Mobile Robots.

Rosenbloom, P.S. and Aasman J. (August, 1990) Knowledge level and inductive uses of chunking (EBL). Proceedings of the National Conference on Artificial Intelligence. , pp. 821-827.

Rosenbloom, P. S., Laird, J. E., Newell, A., and McCarl, R. (1991) "A preliminary analysis of the Soar architecture as a basis for general intelligence". *Artificial Intelligence 47*, 1-3 , 289-325.

Rosenbloom, P., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Lehman, J. F., Rubinoff, R., Schwamb, K., and Tambe, M. (1994) Intelligent Automated Agents for Tactical Air Simulation: A Progress Report. Proceedings of the Conference on Computer Generated Forces and Behavioral Representation.

Rubinoff, R., and Lehman, J. (1994) Natural language processing in an IFOR pilot. Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation.

Stobie, I., Tambe, M., and Rosenbloom, P. (Novemeber, 1992) Flexible integration of path-planning capabilities. Proceedings of the SPIE conference on Mobile Robots.

Tambe, M., and Rosenbloom, P. (July, 1993) On the Masking Effect. Proceedings of the National Conference on Artificial Intelligence.

Tambe, M., Johnson, W. L., Jones, R., Koss, F., Laird, J. E., Rosenbloom, P. S., and Schwamb, K. (Spring 1995) "Intelligent agents for interactive simulation environments". *AI Magazine 16* .

Tan, J. Flying NOE in ModSAF. Private communication.

8. Authors' Biographies

Milind Tambe is a research computer scientist at the Information Sciences Institute, University of Southern California (USC) and a research assistant professor with the computer science department at USC. He completed his undergraduate education in computer science from the Birla Institute of Technology and Science, India in 1986. He received his Ph.D. in computer science from Carnegie Mellon University in 1991. His interests are in the areas of integrated AI systems, agent modeling, plan recognition, and efficiency and scalability of AI programs, especially rule-based systems.

Karl Schwamb is a Programmer Analyst on the Soar Intelligent FORces project at the University of Southern California's Information Sciences Institute. He contributes to the maintenance of the Soar/ModSAF interface software and the Tcl/Tk interface to Soar. He received his M.S. in Computer Science from George Washington University.

Paul S. Rosenbloom is an associate professor of computer science at the University of Southern California and the acting deputy director of the Intelligent Systems Division at the Information Sciences Institute. He received his B.S. degree in mathematical sciences from Stanford University in 1976 and his M.S. and Ph.D. degrees in computer science from Carnegie-Mellon University in 1978 and 1983, respectively. His research centers on integrated intelligent systems (in particular, Soar), but also covers other areas such as machine learning, production systems, planning, and cognitive modeling. He is a Councillor and Fellow of the AAAI and a past Chair of ACM SIGART.